LBL-10690 c2
EEB-BEAG-80-1

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

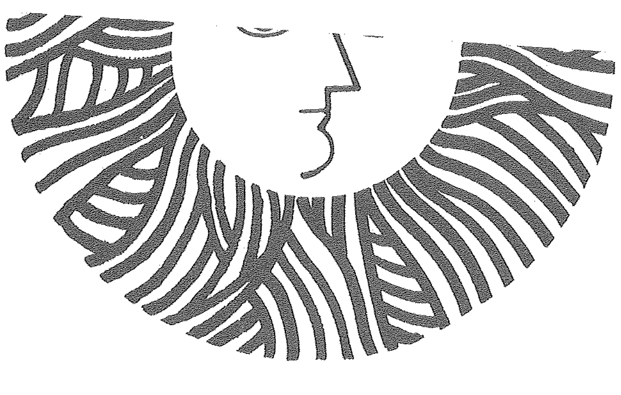
ENERGY & ENVIRONMENT DIVISION

MANUAL FOR UNIX USERS

PLOT: A UNIX PROGRAM FOR INCLUDING GRAPHICS IN DOCUMENTS

Pavel Curtis

April 1980



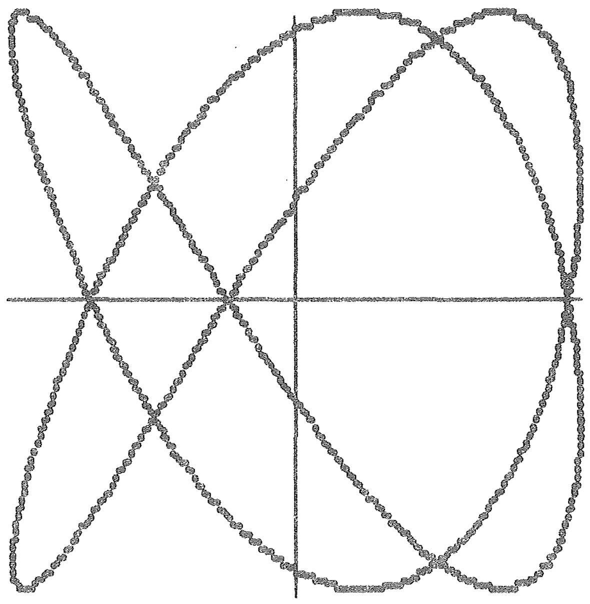
LBL-10690 c.2

Prepared for the U.S. Department of Energy under Contract W-7405-ENG-48

LBL-10690
EEB-BEAG-80-1

Plot: A UNIX[†] Program for Including Graphics in Documents

Pavel Curtis
Building Energy Analysis Group
Lawrence Berkeley Laboratory
April 2, 1980



[†]UNIX is a Trademark of Bell Laboratories.

Plot: A UNIX[†] Program for Including Graphics in Documents

Pavel Curtis
Building Energy Analysis Group
Lawrence Berkeley Laboratory
April 2, 1980

ABSTRACT

Plot is a preprocessor to the document formatting program, nroff, which makes it possible to include many kinds of plots, graphs, and drawings in a document directly, thereby eliminating the necessity, in many cases, for having such illustrations done externally. One also avoids the messy procedure of making sure that nroff leaves enough room for these externally-done plots, pasting it down and possibly unpasting it to move it elsewhere in the document.

Instructions to plot are in a simple, easy-to-read graphics language designed specifically for this application. If desired, the user has complete control over the size, positioning, range and content of the plot. If a great deal of specificity is not required, plot provides reasonable defaults for most parameters, thereby decreasing the number of required input statements. Plots are made using the full capabilities of the target terminal to a greater extent than nroff has done before. This ability, however, limits it to use on certain hardcopy terminals.

[†]UNIX is a Trademark of Bell Laboratories.

Table of Contents

I. INTRODUCTION

Plot translates a simple description of a plot into a complex set of instructions to nroff and the target terminal to produce that plot. Only those portions of the input document which are intended as plot descriptions are looked at and changed by the plot program; all of the rest of the document is transparently passed through. Thus, plot is usable in a pipeline along with tbl and neqn. NOTE, however, that because of an algorithms employed in the plot program, plot must be the first member of any such pipeline. To format a document containing input for both nroff and plot, therefore, a command like the following should be used:

```
plot filename | neqn | tbl | nroff
```

The syntax of a plot command from UNIX is as follows:

```
plot [-s] [-T<terminal>] file ...
```

The -s option is used when using the plot program on files which should not be formatted by nroff. This is primarily intended for those times when only a plot is being run, but can be used anytime nroff formatting is not desired. NOTE that when the -s option is in effect, the ".PS" and ".PE" lines (see below) will not be printed. When the -s option is used, it is not necessary to pipe the output through any of neqn, tbl or nroff. A command like the following will suffice:

```
plot -s filename
```

The -T option is used to specify output for a terminal other than the aj832, which is the default. The only other terminal currently available is the dtc382. Output for the dtc is requested with the command

```
plot -Tdtc file ...
```

For plot to work, an aj832 must be equipped with Ultraplot mode and a dtc382 with Superplot mode. If you are unsure as to whether your terminal is so equipped, you should consult the service representative associated with your terminal.

At the present time, nroff must be called with one of these two forms:

```
When using an aj832:   nroff -Tnewaj -meeb
When using a dtc382:   nroff -T382cw-12 -meeb
```

The 'newaj' and '382cw-12' drivers are the only ones currently on the BKY UNIX system which are usable with the plot program. Garbage output is guaranteed with any other driver in use.

The -meeb macro-package is currently the only package which contains definitions of the ".PS" and ".PE" macros. If you would like to obtain copies of these definitions to add them to another macro-set, address mail to login-name 'beag'. The -meeb macro-package is, for the most part, upwardly-compatible with the -ms macro-package and most things that ran with -ms before, should also run with -meeb. Report any problems to login-name 'beag'.

A plot description in a document must be bounded by ".PS" (Plot Start) and ".PE" (Plot End) macros. Plot processes the intervening instructions but leaves the rest of the document unchanged.

The syntax of the ".PS" command is as follows:

```
.PS <x-width> <y-length> [<indent>]
```

where <x-width> and <y-length> are the desired dimensions of the logical view surface* in inches, and <indent> is an optional parameter specifying the indentation of the plot in inches from the current indent. If an <indent> is given and/or the <x-width> is longer than the current line-length, no checking or truncation is performed. If an <indent> is not given, the plot will be centered in the currently remaining line-length.

NOTE that while <indent> may be given as a number register, both <x-width> and <y-length> must be literally, numbers (although not necessarily integers).

The only other nroff command which may be used between the ".PS" and ".PE" commands is ".so":

.so <filename>	Indicates that said file's contents are to be read and inserted at this point in the input. This is primarily used for modularity and to simplify the task of specifying several similar or identical plots. NOTE that it is an error to end a plot description inside a .so'ed file. A ".PE" may only occur within the original file. NOTE that a .so may not appear in the middle of a set or function definition. As many .so's as desired may appear in a plot description. They may also be nested up to five deep.

*The terms 'logical view surface' and 'viewport' are used in this document with the same meanings as in the GRAFPAC graphics system. Definitions and illustrations appear in the description of the viewport statement in Section II.B.

I. INTRODUCTION

Within the bounds of a ".PS"-".PE" pair, there are three main sections:

1) <u>The Axes Definition Section</u> -- This is the place where the ranges of the x and y coordinate axes, the methods of marking and labeling the axes, etc. are to be specified.

2) <u>The Data Definition Section</u> -- This is the area in which data-point-sets and functions are described.

3) <u>The Drawing Section</u> -- Wherein it is specified which of the above-defined objects are to plotted, in which plot-mode, and with which character or style of line. Also, any text which is to be printed on the plot is herein mentioned.

II. THE AXES DEFINITION SECTION

II.A. The range statement

The range of x and y values for which a plot will be produced may be specified using the range statement. The syntax is as follows:

```
range [x=x1:x2] [y=y1:y2]
```

```
Examples:   range x=-500:500  y=-10:10
            range x=500:-500  y=-10:10
            range y=-10:10
```

The first example states that the x axis should extend from -500 on the left to 500 on the right. Similarly, the y axis will extend from -10 at the bottom of the plot to 10 at the top.

If, as in the second example, the ranges for an axis are in reverse numerical order, then that axis will appear with the normal positive and negative directions also reversed. Thus, in the second example, the x-axis would have 500 as its left limit and -500 on the right.

As the third example illustrates, either one or both of the ranges may be left off entirely. In this case, or if no range statement is found at all, plot will attempt to determine 'good' values, based upon the rest of the plot description. A description of the algorithms used to apply defaults in this and other statements is given in Appendix D.

The question arises as to where to draw the two axes crossing or meeting each other. The following algorithm is used:

1) If an axis covers a range which includes the value zero, then that axis will be crossed at that point.

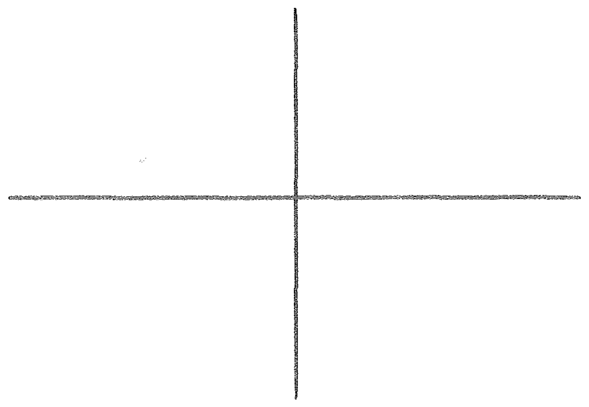
2) If an axis covers a range which includes only positive values, then that axis will be crossed at whichever end represents the lowest value.

3) If an axis covers a range which includes only negative values, then that axis will be crossed at whichever end represents the highest value.

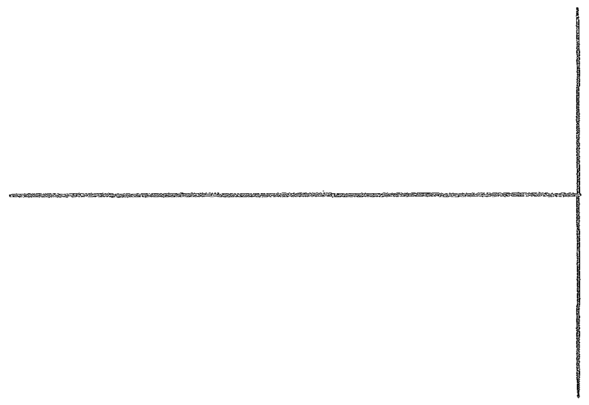
Examples of this algorithm in use appear on the following pages.

II. THE AXES DEFINITION SECTION

SAMPLE PLOTS

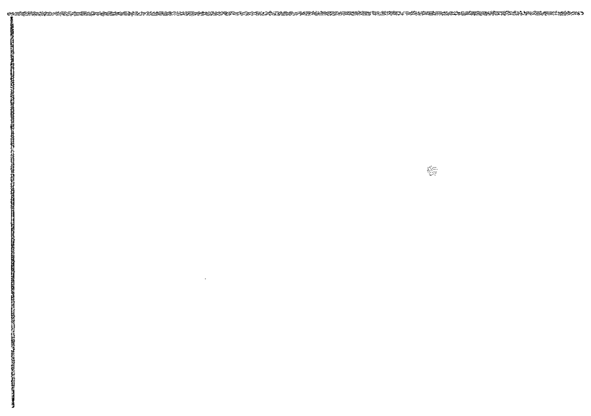


```
.PS 3 2
range x=-500:500 y=-10:10
.PE
```

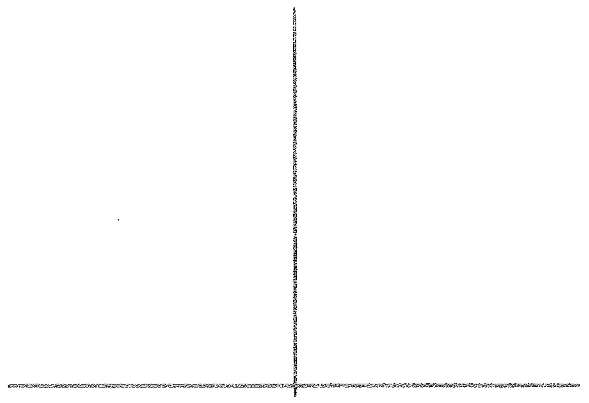


```
.PS 3 2
range x=-500:-100 y=-10:10
.PE
```

SAMPLE PLOTS



```
.PS 3 2
range x=100:500 y=-10:0
.PE
```



```
.PS 3 2
range x=-500:500 y=0:10
.PE
```

II.B. The viewport statement

Plot allows the user to specify a viewport within the logical view surface*. In this way, the user has complete control, should it be desired, of the margins on all sides of the plot. The size of the viewport is given as a pair of points, representing the lower-left- and upper-right-hand corners of the viewport. The syntax is as follows:

```
viewport (<xlowlft>, <ylowlft>) (<xuprght>, <yuprght>)
```

The coordinate system to be used in a viewport statement is based upon the size given for the logical view surface (LVS). Plot considers the largest square which can fit into the logical view surface to have a side of 1 in the LVS coordinate system. Thus, if the ".PS" command for a plot description was ".PS 5 3" then a distance of 1 in the LVS coordinate system for that plot would be equivalent to three inches. If a margin of 1/2 inch on all sides were desired, the viewport statement for the plot would be

```
viewport (.166, .166) (1.5, .833)
```

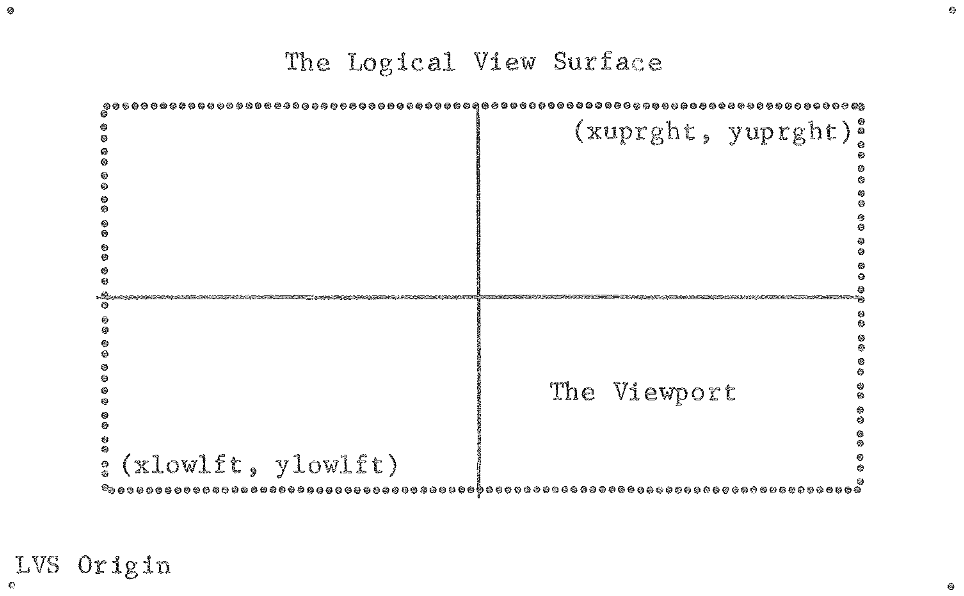
NOTE that the origin of the LVS coordinate system is in the lower-left-hand corner of the logical view surface.

NOTE also that if (and only if) no viewport statement is found, plot will automatically adjust the boundaries to assure that any axis-labels or numbering will appear within the logical view surface.

*The size of the logical view surface is given on the ".PS" command. See Section I.

In the illustration below, the logical view surface is the entire area within the solid line. Note that it is five inches by three inches, the size given in the ".PS" command. The viewport is the smaller sub-area within the logical view surface which is bounded by the dotted line. Note that the axes only stretch to the edges of the viewport, not the logical view surface. That is the definition of the viewport.

The points whose coordinates are given in the viewport are marked by the two ordered pairs in the illustration. The origin of the Logical View Surface coordinate system is also marked.



```
.PS 5 3
viewport (.166, .166) (1.5, .833)
.PE
```

II.C. The marking statement

The x and y axes may each be marked and numbered at arbitrary increments to make the plot easier to read. The marking statement is used to specify if, how, and where each axis is to be marked. The syntax is as follows:

```
marking <axes> [<increment>] [options]
```

<axes> is a one- or two-letter 'word' giving the names of the axes being referred to in this statement. The only possible values are 'x', 'y' and 'xy'.

<increment> is an optional parameter specifying how often the axis (axes) will be marked. If a value is given, the axis will be marked every <increment> units. If no value is given, the axis will be marked every inch.

NOTE that if the value at the end of an axis is a multiple of the <increment>, that position will not be marked. The value zero, if present, is never marked. The ends may be marked by giving a slightly wider range to that axis. For example, given the statements

```
range x=-500:500
marking x 100
```

the ends of the x-axis will not be marked, whereas if the following statements were used:

```
range x=-510:510
marking x 100
```

the x-axis would be marked very near the ends.

The options available on the marking statement are the following:

nohash	requests that no hash-marks (short perpendicular lines crossing the axis) be drawn for this axis. If nohash is not specified, then hash-marks and numbers (see nonumber, below) will be printed every <increment> units.
nonumber	requests that no numbers (representing the value of the axis at a certain point) will be printed for this axis. NOTE that if both nohash and nonumber are given for an axis, that axis will appear completely unmarked. This same effect could more easily be achieved by simply not including that axis in any marking statement at all.
left right	specify where the numbers for the y-axis will be printed (i.e. to the left or right of the axis). The default positioning is right.

top bottom	specify where the numbers for the x-axis will be printed (i.e. above or below the axis). The default positioning is <u>bottom</u>.

```
Examples:   marking xy 25

            marking x 10 nonumber
            marking y 40 nohash

            marking x
            marking y 30 nonumber nohash
```

If both axes are to be marked in exactly the same fashion, the specification need only be given once, as in the first example, which states that both axes are to be hashed and numbered every 25 units. If the axes are to be marked in different fashions, a <u>marking</u> statement must appear for each of the axes, as in the other examples.

The second example states that the x-axis will be hashed every 10 units but not numbered and that the y-axis is to be numbered every 40 units, but not hashed.

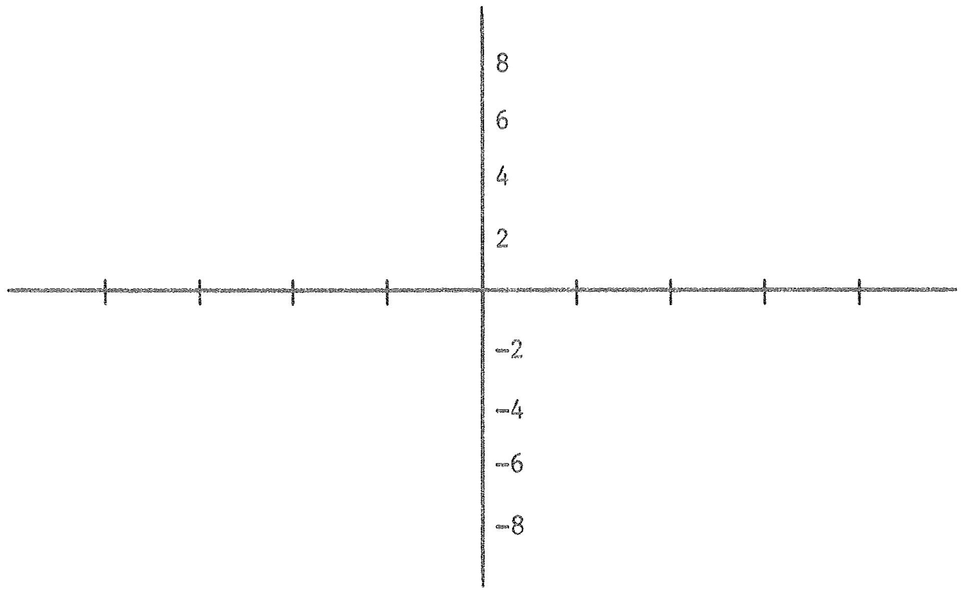
The third example shows a number of interesting points. For the x-axis, both hashing and numbering are to occur, but no increment is given. In such a case, plot will set the increment to whatever number of units will be scaled into an inch.

For the y-axis, no hashing or numbering is to be done at all. That is to say, the y-axis will be unmarked.

<u>NOTE</u> that if both <u>nohash</u> and <u>nonumber</u> are given, any increment will be ignored. <u>NOTE</u> also that the same effect (i.e. an unmarked axis) can be achieved by simply not including that axis in any <u>marking</u> statement.

II. THE AXES DEFINITION SECTION

SAMPLE PLOT



```
.PS 5 3
range x=-500:500 y=-10:10
marking x 100 nonumber
marking y 2 nohash
.PE
```

II.D. The label statement

Each axis may have a label, usually describing its function, written next to it. This is done with the label statement, the syntax of which is as follows:

```
label <axes> "<label>"
```

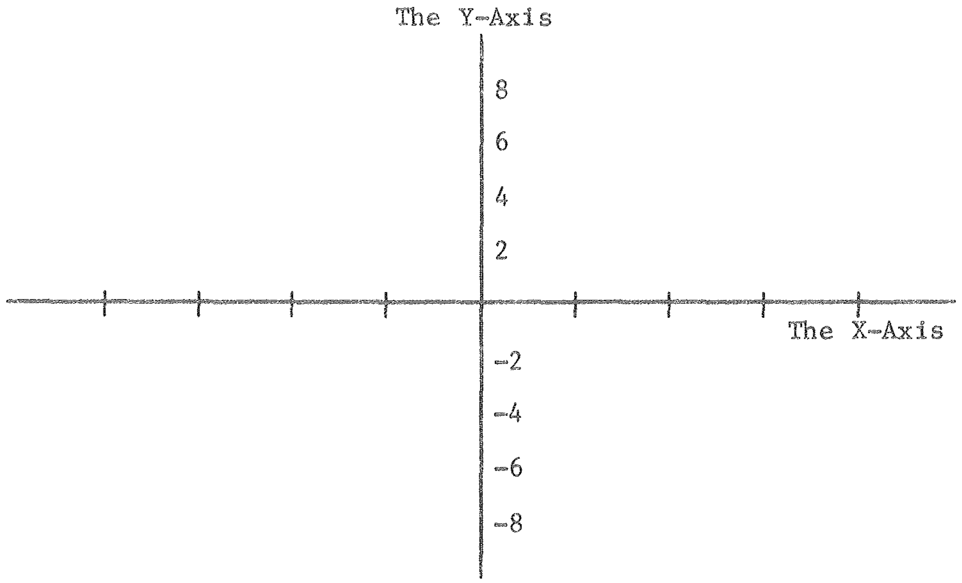
```
Examples:  label xy  "Coordinate axis"

           label x "Number of Users Logged-On"
           label y "Time to do Trivial C Compilation"
```

NOTE that labels for the x-axis will appear just below the axis and right justifed in the viewport. Those for the y-axis will be printed centered above the y-axis or justified in the viewport, if necessary. If a different placement is required, the write statement should be used. (The description of the write statement appears in Section IV.D.)

NOTE also that the labels (and, incidentally, text given to the write statement) are printed using the normal horizontal pitch of the terminal. Plot does no checking to be certain that the text given will necessarily fit in the logical view surface.

II. THE AXES DEFINITION SECTION

SAMPLE PLOT



```
.PS 5 3
range x=-500:500 y=-10:10
marking x 100 nonumber
marking y 2 nohash
label x "The X-Axis"
label y "The Y-Axis"
.PE
```

III. THE DATA AND FUNCTION DEFINITION SECTION

III.A. Defining Sets of Data-Points

The syntax for a definition of a set of data-points is as follows:

```
          define <set-name> : (<x-coord>, <y-coord>) ...
                              (<x-coord>, <y-coord>) ...
                                         .
                                         .
                                         .
```

The first line of the definition begins with the keyword define, which is followed by the name of the set. Set-names have a letter as the first character followed by an unbroken series of letters, digits and/or underlines. No other characters may be imbedded in a data-set name.

NOTE that only the first 10 characters of a data-set name are significant. Any characters above that limit will be ignored.

After the set-name must be a colon followed by a list of ordered pairs, each pair separated from the others, if desired, by spaces, tabs and/or newlines. Each ordered pair may have embedded blanks (e.g. after the comma) if desired.

The extent of a set definition is defined as all ordered pairs found between the set-name and the next plot keyword.

A set of data-points will only be plotted if a plot statement appears for that set. (See Section IV.B.)

```
Examples:   define Critical_Points : (45, 3) (21, 21) (-3, -6) (0, 0)
                                     (8, 80) (-5, -51) (0, 2) (56, -1)
            define Another_Set : (0, 0) (-1, -1) (-1, 0) (0, -1) (0, 0)
```

III.B. Defining Functions

Functions may be defined in one of three fashions:

1) By Expression — The function is simple enough to be stated in a single FORTRAN expression.

2) By Algorithm — The function is defined in a fashion similar to that of a FORTRAN function, using a small FORTRAN-like language.

3) Parametrically — The x and y coordinates are each governed by separate functions based upon a common independent variable, t.

We will look at each alternative separately.

III.B.1 Function-Definition By Expression

By far the simplest method of defining a function, the syntax is as follows:

```
define <function-name> = <expression> ;
```

A function-name consists of an identifier, which is a letter followed by an unbroken string of letters, digits and/or underlines. The identifier is followed by a left parenthesis, the independent variable (which must be either x or y), and a right parenthesis. The following are all legal function-names:

```
quad(x)    s45(y)    left_hand_differential(x)    f(y)
```

NOTE, however, that only the first 10 characters of any identifier are significant. Thus, if a function left_hand_integral(y) existed in addition to all those above, an error would surely result, since it and left_hand_differential(x) have the same first 10 characters.

Expressions have exactly the same syntax as standard FORTRAN expressions. They may continue over the ends of lines so long as no identifier, keyword or constant is split up. The end of the expression is determined by the trailing semicolon.

The following mathematical functions may be used without definition:

sin(x)	
cos(x)	
tan(x)	
atan(x)	The standard trigonometric functions, these deal in radians only. The returned value from atan(x) will be in the range $-\frac{\pi}{2} \leq$ atan(x) $\leq \frac{\pi}{2}$.

exp(x)
log(x)
log10(x)
	log(x) is the log base e. Both log(x) and log10(x) return a very large negative number at x=0. It is an error to take the log of a negative number.

gamma(x)
sqrt(x)
abs(x)
	The $\Gamma(x)$, square root and absolute-value functions. It is an error to take the square root of a negative number.

mod(x, y)	Returns a value f, such that x = iy + f, where i is an integer and $0 \le f < y$.

floor(x)
ceil(x)
	floor(x) returns the largest integer not greater than x. ceil(x) returns the smallest integer not less than x.

In addition, any user-defined function may be used if its definition appeared earlier in the file.

EXAMPLES

```
define cube(x) = x ** 3;

define quadratic(y) = 5 * y**2 + 6 * y + 4;

define phase(x) = sin(x) * sin(20.5 * x);

define unl(x) = .5 + .1 * x + .4 * x**2;

define power(y) = y ** y;

define singularity(x) = 1 / (x**2 - 1);
```

III.B.2 Function-Definition By Algorithm

It is often the case that a function can not be stated as a single expression. In order to overcome this difficulty, a small, FORTRAN-like language called PLOTRAN has been created. Using PLOTRAN, it is possible to define almost all single-valued functions. An example of a function defined in PLOTRAN is as follows (The numbers in parentheses are for reference only and would not appear in a real definition):

```
 (1)   define sample(x):
 (2)            prod = 1;
 (3)            if (x < 0) goto 10;
 (4)            for k=1 to x;
 (5)                 prod = prod * 2;
 (6)            next k;
 (7)            goto 20;
 (8)        10: for k=-1 to x step -1;
 (9)                 prod = prod / 2;
(10)            next k;
(11)        20: sample = prod;
(12)            end;
```

Line (1) corresponds to the <u>define</u> statement in a function-definition by expression. In this case, however, we do not have a single expression, so we end the statement with a colon.

Line (2) is an example of a PLOTRAN assignment statement. It computes a value from the expression on the right side of the equals sign and stores it in the variable named on the left side.

PLOTRAN variable names have exactly the same form as a data-point-set name, that is, a letter followed by an unbroken string of letters, digits, and/or underlines. <u>NOTE</u> that in PLOTRAN, as with data-point-sets, only the first 10 characters of a variable name are significant. More characters may be present, but they will be ignored by the plot program.

In any function definition, three variables have a special meaning to plot. The independent variable ('x' in the example above) is set upon entry to the function. The dependent variable, which is always the one of x or y which is not the independent variable (e.g. 'y' would be the dependent variable in the example above), and the variable which has the same name as the function (i.e. 'sample' in the example above) refer to the same space in memory. Thus if one of the two is changed, the other is, of course, also changed.

<u>NOTE</u> that all variables other than the independent variable are set to zero upon entry to the function.

NOTE carefully the trailing semicolon on this and all subsequent lines. Excepting only the define statement, all PLOTRAN statements must end with a semicolon. This feature allows statements to be carried over line boundaries, so long as no constant, keyword or name is split up.

Line (3) illustrates the PLOTRAN if statement, the general syntax of which is as follows:

```
if ( <expression> ) goto <statement-number> ;
```

The expression given in an if or assignment statement may use any of the operators commonly used in FORTRAN (i.e. +, -, *, / and **), any of the usual comparison operators (i.e. >, <, =, >=, <=, and <>, which means 'not equal'), and the Boolean operators & (and), | (or) and ! (not).

The expression

```
A > B
```

will have the value 1 (one) if the value of the variable A is numerically greater than that of the variable B. It will have the value 0 (zero) otherwise. This pattern is followed with all of the comparison operators.

In a similar manner, the expression

```
A & B
```

will have the value 1 (one) if both A and B are nonzero and the value 0 (zero) otherwise. The ! (not) operator, when applied to an expression, gives the value 1 (one) if the expression equals zero and 0 (zero) otherwise.

Given this set of rules, we can now understand the workings of the PLOTRAN if statement.

When executing an if statement, plot evaluates the expression within the parentheses. If the value of the expression is nonzero, then control will transfer to the statement within the function-definition which has the number given after the goto keyword. In the case of the if statement in line (3), control would be transfered to line (8), since it has the label '10'.

Lines (4) through (6) demonstrate the PLOTRAN for-next construction. This is used when a number of statements are to be executed several times. The syntax for the two statements is as follows:

```
for <variable> = <expression> to <expression> ;
                   OR
for <variable> = <expression> to <expression> step <constant> ;
      .
      .
      .
next <variable> ;
```

Using the first form of the for statement is exactly equivalent to using the second form with the <constant> equal to 1 (one). From this point on, we shall refer to that <constant> as the 'stepsize'. The effect of a for-next pair is exactly as follows:

1) The variable is assigned the value of the first expression.

2) If the stepsize is positive and the value of the variable is greater than that of the second expression, OR the stepsize is negative and the value of the variable is less than that of the second expression, control will skip to the statement directly after the corresponding next statement.

3) All of the statements between the for and the next will be executed.

4) The value of the stepsize will be added to that of the variable and the result stored back in the variable.

5) Steps 2) through 4) are repeated until the conditions stated in step 2) are met.

Thus, the effect of lines (4) through (6) is to double the value of the variable prod 'x' times*, while that of lines (8) through (10) is to halve the value of prod the same number of times.

The only lines left to be explained are (7), (11) and (12). Lines (7) and (11) are, of course, examples of the PLOTRAN goto statement, the effect of which is to unconditionally transfer control to the indicated statement. Line (12), the end statement, does just that. Whenever control reaches an end statement, execution halts and the value of the dependent variable is returned from the function.

NOTE that only one end statement may appear in any function and that it must occur as the very last statement in the definition. The end statement is plot's only way of knowing when a function-definition is completed.

*Actually, the loop will be executed a number of times equal to the integer part of 'x'.

A summary of all PLOTRAN statements and their syntax is given in Appendix B. An explanation of all PLOTRAN operators and their relative precedence is given in Appendix C.

The following is another example of a PLOTRAN function-definition. Further examples may be found in the Sample Plot section, Appendix E.

```
define root(y) :
          if (y < 0) goto 40;
          oldrt = 2;
      10: newrt = (oldrt + y / oldrt) / 2;
          if (oldrt > newrt) goto 20;
          if (newrt - oldrt < 1e-6) goto 30;
          oldrt = newrt;
          goto 10;
      20: if (oldrt - newrt < 1e-6) goto 30;
          oldrt = newrt;
          goto 10;
      30: root(y) = newrt;
          goto 99;
      40: root(y) = 0;
      99: end;
```

III.B.3 Parametric Function-Definitions

A great number of functions (primarily, but not solely, those which are multiple-valued at some or all points) are not definable in either of the styles given above. To attempt to partially overcome this deficiency, the facility is given for defining a function parametrically. Such a definition takes the following form:

```
define <parametric-function-name> :
          definition of x>
         <definition of y>
```

Parametric-function-names are exactly like other function-names, except that the independent variable is <u>always</u> ′t′. The definitions of x and y are in either of the two forms given above, excepting the ′define′ keyword, which <u>must</u> be omitted.

<u>NOTE</u> that the definitions of x and y <u>must</u> each start a line.

An example of a parametric function-defintion is as follows:

```
define circle(t) :
       x = 3 * cos(t);
       y
```

π, would produce a circle, assuming that the increment were small enough.

See Section IV.B. for an explanation of how parametric functions are plotted.

IV. THE DRAWING SECTION

IV.A. The Coordinate Axes

Unless told not to, the plot program will draw both coordinate axes on the graph. To prevent this occurance, the noplot statement must be employed. The syntax is as follows:

```
                    noplot <axes>

     Examples:      noplot xy
                    noplot x
                    noplot y
```

The first example asks that neither axis be drawn. This does not mean that any numbers, marks, or labels requested will not be drawn. If a marking or label statement appeared in the axis-definition section, the instructions found there will be obeyed, regardless of the presence or absence of a noplot statement.

The second example asks that the x-axis not be drawn, and the third does the same for the y-axis.

IV.B. The plot statement

The plot program will only draw graphs of those functions and data-point-sets that are specifically requested. The request to draw appears in one of the follwing forms, discussed below.

IV.B.1. Data-Point-Sets

A request to plot the points in a data-set appears as follows:

```
plot <set-name> ["<character>"]
```

<set-name> is the name of a data-set defined earlier in the plot description. <character> is the printing character that will be used to plot the points in the set. The default plotting character is a plus ('+').

IV.B.2. Connected Points

If a straight line is desired connecting the points of a data-set, this form of the plot statement should be used:

```
plot connect <set-name> [<line-type>]
```

<set-name> is the name of a data-set defined earlier in the plot description. <line-type> specifies what kind of line should be used (i.e. solid, dotted, dashed, etc.). Line-types are defined more completely in Section IV.B.5.

NOTE that the points will be connected in the order in which they were specified in that set's define statement.

IV.B.3. Functions

The request to plot a function appears as follows:

```
plot <function-name> [<range>] [<line-type>]
```

<function-name> is the name of a function defined earlier in the plot description. NOTE that the indpendent variable given in the plot statement need not be the same as that given in the corresponding define statement. This allows a function to be plotted using either or both of x and y as the independent variable, as in the following sequence:

```
define root(x) = sqrt(x);
plot root(x)
plot root(y)
```

Ordinarily, a given function is plotted over the entire range of its independent variable. If this is not desired, a range for the independent variable may be given. The syntax of a range is exactly like that of the range statement (See Section II.A.), that is,

```
variable> = <begin> : <end>
```

where the <variable> must be the same as the independent variable given earlier on the same line. An example of a plot statement using a range is as follows:

```
plot root(x) x=0:100
```

<line-type> specifies what kind of line should be used when plotting the function (i.e. solid, dotted, dashed, etc.). Line-types are defined more completely in Section IV.B.5.

IV.B.4. Parametric Functions

The request to draw a parametric function is much like that of a normal function. The syntax is as follows:

```
plot <parametric-function-name> [<range>] [dt=<increment>] [<line-type]
```

parametric-function-name> is the name of a parametric function defined earlier in the plot description. <range> is the range of values that t will take on during the plotting of the function (see below). The syntax of a range specification is exactly as given in Section IV.B.3. above.

<increment> is defined below.

<line-type> specifies what kind of line should be used when plotting the function (i.e. solid, dotted, dashed, etc.). Line-types are defined more completely in Section IV.B.5.

When plotting a parametric function, certain exceptions apply:

1) If no range is given, the range will be that of the x-axis.

2) The increment for the value of t (see below) is given in the plot statement in the form

   dt=<increment>

3) If no increment is given, it will default to the smallest increment on the x-axis which is representable, given the precision of the output terminal.

A parametric function is plotted in the following manner:

1) An intial value for t is found (see above).

2) Using the definitions of x and y, a coordinate pair is constructed.

3) The pen moves to this initial position.

4) The value of t is increased be one <increment> and the new value checked to see if it exceeds the final value given for t. If so, the plot is concluded. Otherwise...

5) A new coordinate pair is constructed using the new value of t.

6) A straight line is drawn from the pen's current location to the position indicated by the new coordinate-pair.

7) Steps 4) through 6) are repeated until t exceeds its limit.

IV.B.5. Line-types

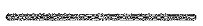
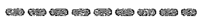
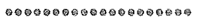
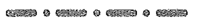
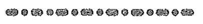
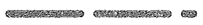
There are three standard types of lines available for functions to be drawn with: solid, dotted, and dashed. To specify one of the standard types, simply give its name in the proper position on the line, as given in that statement's syntax. The default type is solid.

Users may also define their own line-types by specifying a string of at most 20 digits enclosed in double quotes ("). The string will be read one digit at a time, interpreting the first digit as a count of increments along the line with the pen down, the second digit a count with pen up, the third down, fourth up, etc. When plot reaches the end of the string, it will start over again at the beginning. For example:

> The string "5212" means that for the first 5 increments along the line, the pen will be down (i.e. periods will be printed). For the next 2, the pen will be up, then down for 1 and up for 2 more. Then the cycle will begin all over again, 5 down, 2 up, 1 down, 2 up, etc.

NOTE that if a component of a user-defined line needs to be longer than 9 increments, it should be specified as a run of digits separated by zeroes, as in "909064" which means 24 down, 4 up, 24 down, 4 up, etc.

A sample of various line-types is given below:

	This is the default, or 'solid' line.
	This is a 'dashed' line, equivalent to "52".
	This is a 'dotted' line, equivalent to "12".
	This line is defined as "9313".
	This line is defined as "3212".
	This line is defined as "909064".

IV.C. Bar Graphs and Histograms

Once a data-point-set has been defined, those points may be used to draw a histogram or a bar graph. The syntax for such requests are as follows:

```
   plot histogram <set-name> [<line-type>]
   plot bargraph <set-name> [<line-type>] [vertical] [width=<bar-width>]
```

Bargraphs and histograms are essentially the same thing in slightly different formats. Examples of each are given below. NOTE that bar graphs are always drawn horizontally unless the vertical keyword is given. Histograms are always vertical.

For histograms, each point in the data-set will be interpreted as the right-hand corner of a histogram-bar. The only exception is when the second point in the set is to the left of the first. In this case, each point will be interpreted as the left-hand corner of the bar.

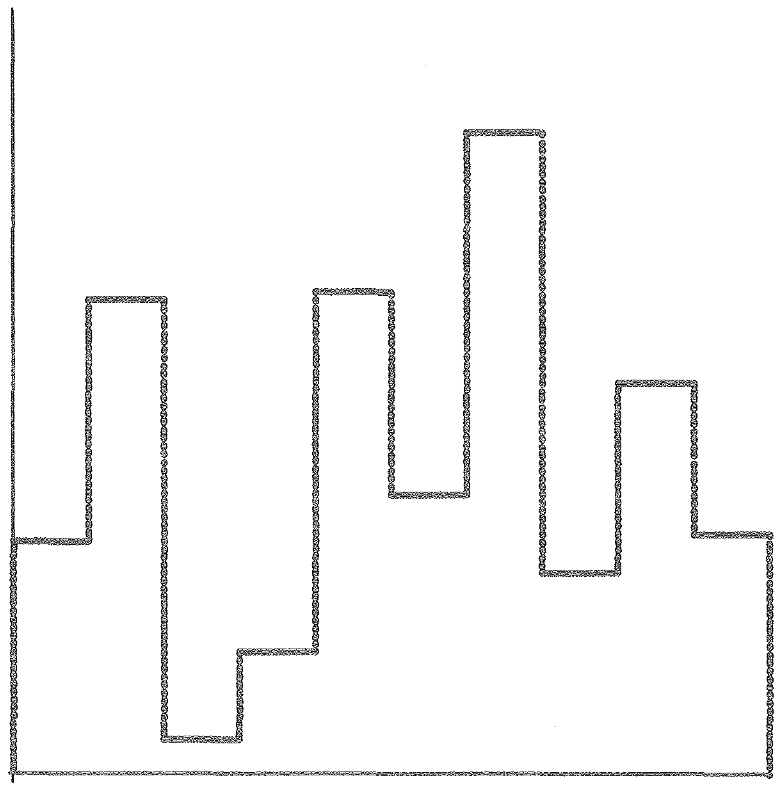
The width of each bar will be equal to the x-distance between the given point and the one before it. The width of the very first bar will be equal to that of the second (i.e. the x-distance between the first and second points).

For bar graphs, the points are always interpreted as the upper-right corner of the bar. <bar-width> is the width of each bar in the same scale as the y-axis. (If the vertical keyword has been given, then the width is interpreted as being in the same scale as the x-axis.)

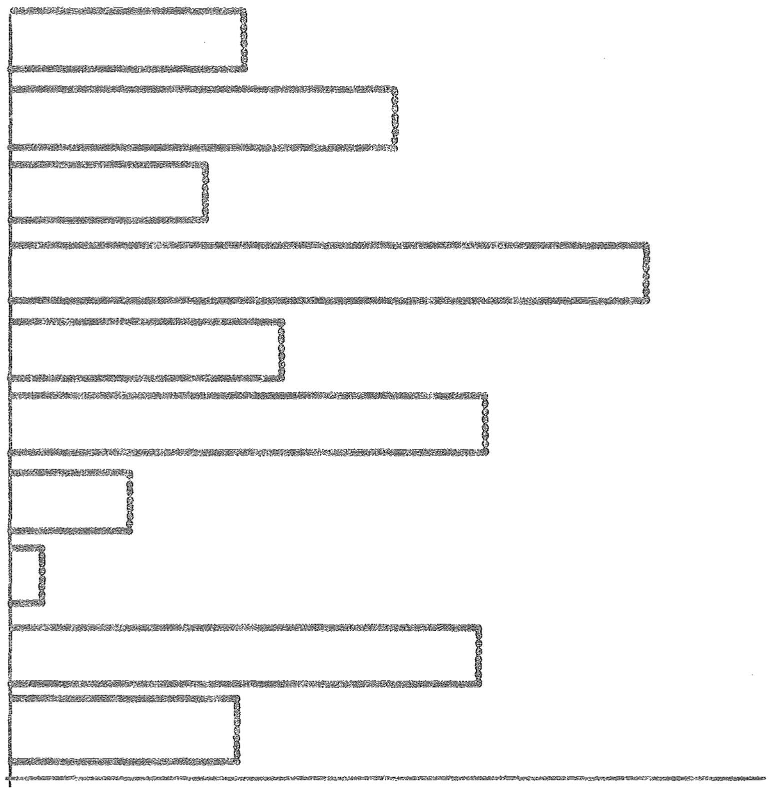
If no width is given in the plot-statement, the width of each bar will be 80% of the y-distance between the given point and the previous one. The width of the very first bar will be equal to that of the second.

For either histograms or bar graphs, a line-type may be specified which will be used in drawing the bars. The syntax of such a request is exactly as given for line-types in function-plots, above.

SAMPLE PLOTS



```
.PS 4 4
range x=0:10 y=0:100
define Sample_hist : (1,30) (2,62) (3,4) (4,16) (5,63) (6,36)
                     (7,84) (8,26) (9,51) (10,31)
plot histogram Sample_hist
.PE
```



```
.PS 4 4
range x=0:100 y=0:10
define Sample_barg : (30,1) (62,2) (4,3) (16,4) (63,5) (36,6)
                          (84,7) (26,8) (51,9) (31,10)
plot bargraph Sample_barg
.PE
```

IV.D. The write statement

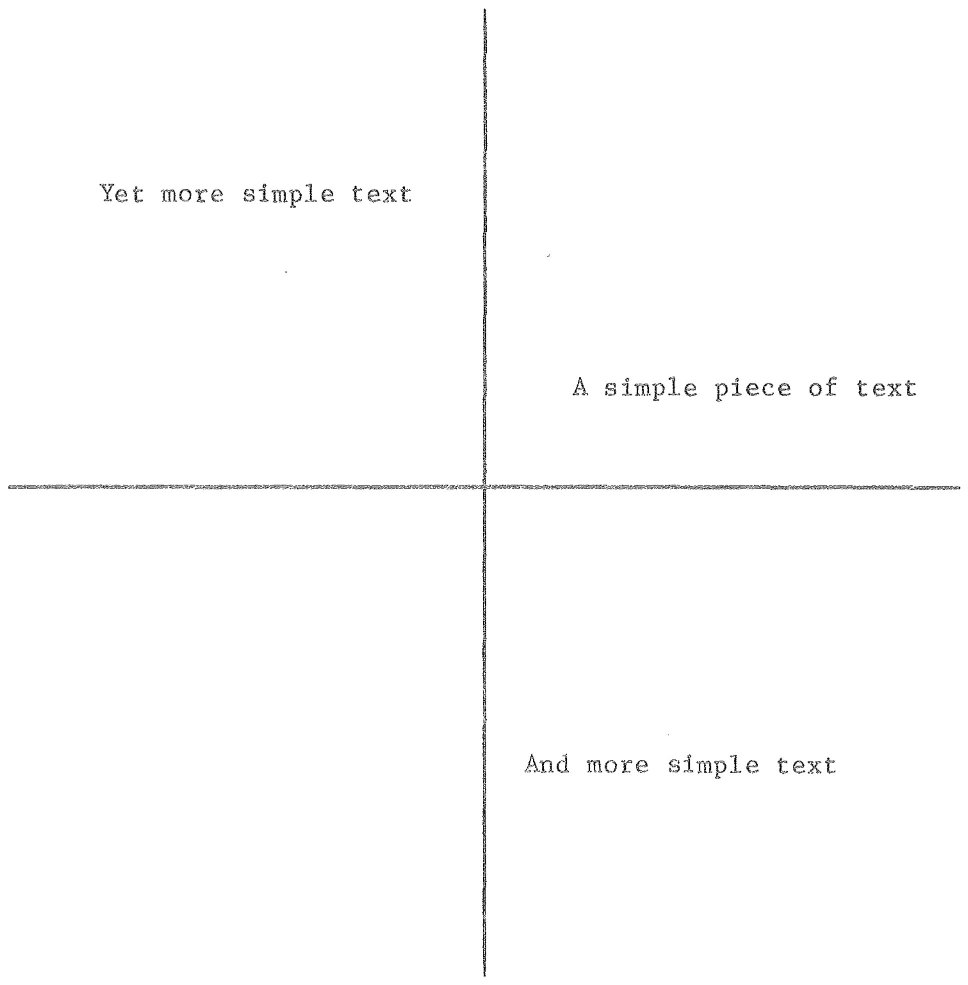
It is often the case that one needs to have text printed somewhere on the plot, be it labels for functions, descriptions of the plot, etc. The write statement provides this facility. The syntax of the write statement is as follows:

```
write (<x>, <y>) "<text>"
write l(<x>, <y>) "<text>"
write L(<x>, <y>) "<text>"
```

The first form is used to specify a location in the same system that your data is in. Either of the other forms may be used to specify a location in LVS coordinates. (LVS coordinates are explained in the description of the viewport statement, Section II.B) The two LVS forms are exactly equivalent; the choice is provided merely for user convenience.

NOTE that the character just before the '(' in the second form is a lower-case letter 'L'.

SAMPLE PLOT



```
.PS 5 5
range x=-10:10 y=-10:10
write (2,2) "A simple piece of text"
write l(.1,.8) "Yet more simple text"
write L(.55,.2) "And more simple text"
.PE
```

IV.E. The Legend

A legend, describing what each graph represents, can also be printed. The syntax for such a request is as follows:

```
legend  (<x>,<y>)   <line-type>  "<description>"
legend  l(<x>,<y>)  <line-type>  "<description>"
legend  L(<x>,<y>)  <line-type>  "<description>"

legend  (<x>,<y>)   '<string>'   "<description>"
legend  l(<x>,<y>)  '<string>'   "<description>"
legend  L(<x>,<y>)  '<string>'   "<description>"
```

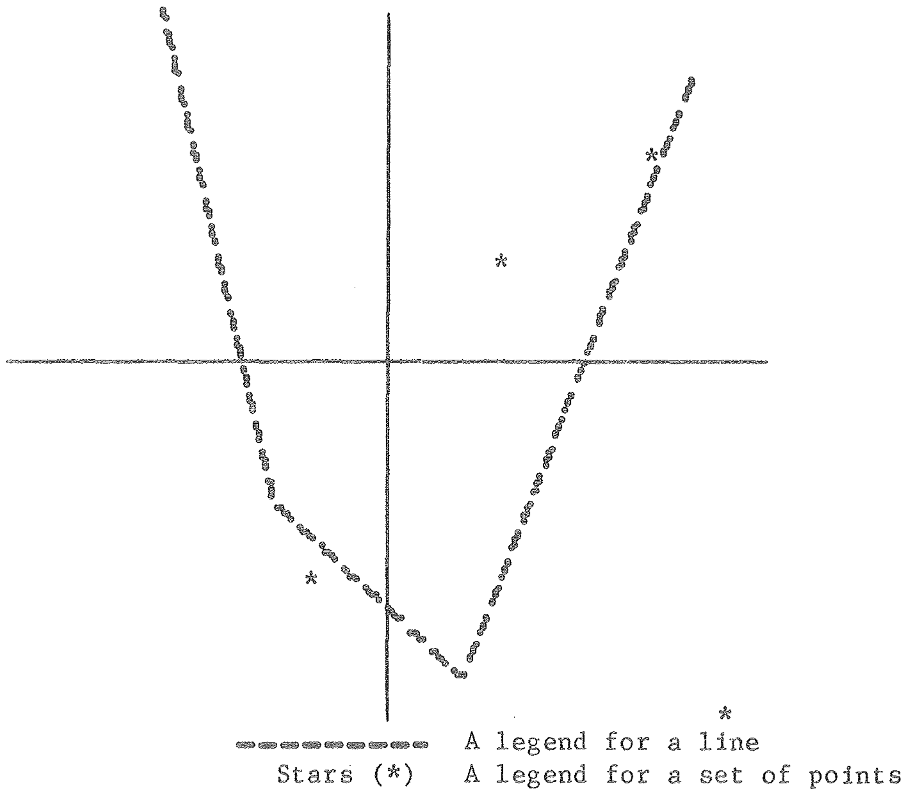
Each legend statement begins with a pair of coordinates (the syntax of which is exactly like that of the coordinates in the write statement) which specify where this legend line will be placed.

For lines, this is followed by a <line-type>, presumably, but not necessarily, the same type as one of the lines in the plot. Exactly one inch of this line type will be printed.

For data-point sets, the coordinates are followed by a string, enclosed in single quotes, which presumably lists the character(s) used to plot one or more data-point sets. The maximum length of this string is 12 characters. It will be printed left-justified in a 12-character field.

In either case, a one-line description of at most 50 characters follows, enclosed in double quotes. This description will be printed 1/4 inch to the right of the line or string.

SAMPLE PLOT



```
.PS 4 4
range x=-10:10 y=-10:10
viewport (0,.0833) (1,1)
define Set_One : (8,8) (2,-9) (-3,-4) (-6,10)
define Set_Two : (7,6) (-2,-6) (3,3) (9,-10)
plot connect Set_One dashed
plot Set_Two "*"
legend L(.3,.0416) dashed "A legend for a line"
legend (.3,0) ´Stars (*)´ "A legend for a set of points"
.PE
```

A. SUMMARY OF PLOT STATEMENTS

Appendix A. Summary of Plot Statements

The ".PS" Command
.PS <x-width> <y-length> [<indent>]

The Axis Definition Section
range [x=x1:x2] [y=y1:y2]
viewport (<xlowlft>, <ylowlft>) (<xuprght>, <yuprght>)
marking <axes> [<increment>] [<options>]
label <axes> "<label>"

The Data and Function Definition Section
define <set-name> : (<x>, <y>) ...
define <function-name> = <expression> ;
define <function-name> : <PLOTRAN-function-definition>
define <parametric-function-name> : <definition of x> <definition of y>

The Drawing Section
noplot <axes>
plot <set-name> ["<character>"]
plot connect <set-name> [<line-type>]
plot <function-name> [<range>] [<line-type>]
plot <parametric-function-name> [<range>] [dt=<increment>] [<line-type>]
plot histogram <set-name> [<line-type>]
plot bargraph <set-name> [width=<bar-width>] [vertical] [<line-type>]
write (<x>, <y>) "<text>" write l(<x>, <y>) "<text>" write L(<x>, <y>) "<text>"
legend (<x>, <y>) <line-type> "<description>" legend l(<x>, <y>) <line-type> "<description>" legend L(<x>, <y>) <line-type> "<description>" legend (<x>, <y>) '<string>' "<description>" legend l(<x>, <y>) '<string>' "<description>" legend L(<x>, <y>) '<string>' "<description>"

Appendix B. A Formal Syntax of PLOTRAN

In the following syntax equations, a modification of Backus Normal Form is used. The following punctuation conventions are followed:

<term>	A term enclosed in angle brackets represents the syntactic entity named by the term.
[anything]	Anything enclosed in square brackets is optional.
anything ...	Anything followed by an elipsis may be repeated z
2	Either one or the other of the choices may be used, but not both at once.
{anything}	Anything enclosed in curly braces should be considered as a unit.

Any word appearing without enclosing punctuation or any character other than the ones mentioned above should be used literally as given. A character mentioned above but appearing in quotes loses its special meaning and should be used as given, without the quotes.

```
 <function-definition> ::= [<statement>] ... <end-statement>

           <statement> ::= [<label>] { <assignment-statement>
                                     | <if-statement>
                                     |  goto-statement>
                                     | <for-statement>
                                     | <next-statement> }

       <end-statement> ::= [<label>] end ;

               <label> ::= <unsigned-integer> :

<assignment-statement> ::= <lvalue> = <expression> ;

              <lvalue> ::= <identifier> [ ( <independent-variable> ) ]

        <if-statement> ::= if ( <expression> ) goto <unsigned-integer> ;

      <goto-statement> ::= goto <unsigned-integer> ;

       <for-statement> ::= for <identifier> = <expression> to
                                        <expression> [step <constant>] ;

      <next-statement> ::= next <identifier> ;
```

B. PLOTRAN SYNTAX

```
  <expression> ::= <expr2> [ {"|"  |  &} <expr2> ] ...

       <expr2> ::= <simple-expr> [<relate-op> <simple-expr>] ...

   <relate-op> ::=  =  |  <>  |  <  |  >  |  <=  |  >=

 <simple-expr> ::= [+ | -] <term> [ {+ | -} <term> ] ...

        <term> ::= <factor> [ {* | /} <factor> ] ...

      <factor> ::= <atom> [ ** <atom> ]

        <atom> ::= <unsigned-constant> |
                          <identifier> |
                       <function-call> |
                      ( <expression> ) |
                              ! <atom> |

  <identifier> ::= <letter> [ <letter> | <digit> | "_" ] ...
```

Appendix C. The PLOTRAN Operators and Their Relative Precedence

This appendix is meant to serve as a reference when it is necessary to know exactly how the PLOTRAN operators behave, what values they return and in what order expressions will be evaluated.

The following is a complete list of the PLOTRAN operators showing their relative precedence, returned value and associativity. Those operators nearer the top of the table have higher precedence and will be evaluated earlier in any given expression. Those operators shown on the same line have identical precedence and will evaluated according to their associativity. Parentheses may always be used to change the order of evaluation.

The PLOTRAN Operators

Operator	Associativity	Value Returned
!	right to left	!X equals one if X=0, zero otherwise.
**	left to right	X**Y equals X^Y. If X<0, Y is truncated to an integer value.
* /	left to right	As in FORTRAN.
unary - + -	left to right	As in FORTRAN.
= <> < > <= >=	left to right	X op Y equals one if the condition is true, zero otherwise.
& |	left to right	X & Y equals one if <u>both</u> X and Y are nonzero, zero otherwise. X | Y equals one if <u>either</u> X or Y is nonzero, zero otherwise.

Appendix D. The plot Program Defaulting Algorithms

In almost all of the plot program statements, parameters or options may be left off by the user. This section describes the actions taken by the plot program in each instance. If a parameter is not mentioned below, then it is an error to leave it off of the corresponding statement.

The range Statement

If no range for a variable is given, one of two things will occur:

1) If any datasets were defined, the range of the variable in question will be the range of values present in the datasets. If that range is of zero width, the range of the variable will be expanded by one in each direction.

2) If no datasets were defined, the range of that variable will be -1:1.

The viewport Statement

If only one ordered pair is given, the other pair will be assumed to have been (1,1).

If no pairs were given or no viewport statement appeared at all, values of xlowlft, ylowlft, etc. will be calculated which adjust for any axis-numbers or labels requested, so as to keep them within the logical view surface.

The marking Statement

If no <increment> is given, an increment equal to one inch will be assumed.

If neither left or right is given and marking of the y-axis is requested, marking will occur on the right.

If neither top or bottom is given and marking of the x-axis is requested, marking will occur on the bottom.

The plot Statement (Datasets)

If no <character> is given, the points will be plotted with a plus ('+').

The plot connect Statement

If no <line-type> is given, solid is assumed.

The plot Statement (Functions)

If no <range> is given, the entire range of the independent variable will be used. If no <line-type> is given, solid is assumed.

The plot Statement (Parametric Functions)

If no <range> is given, the range of the x-axis will be used. If no <increment> is given, the smallest increment on the x-axis, given the precision of the output terminal, will be used.

The plot histogram Statement

If no <line-type> is given, solid is assumed.

The plot bargraph Statement

If no <line-type> is given, solid is assumed. Unless the vertical keyword is given, the bargraph will be drawn horizontally. If no <width> is given, the width will be 80% of the y-distance (x-distance if vertical) between the first and second points. If there is no second point, the width will be equal to 1/2 inch.

Appendix E. Sample Plots

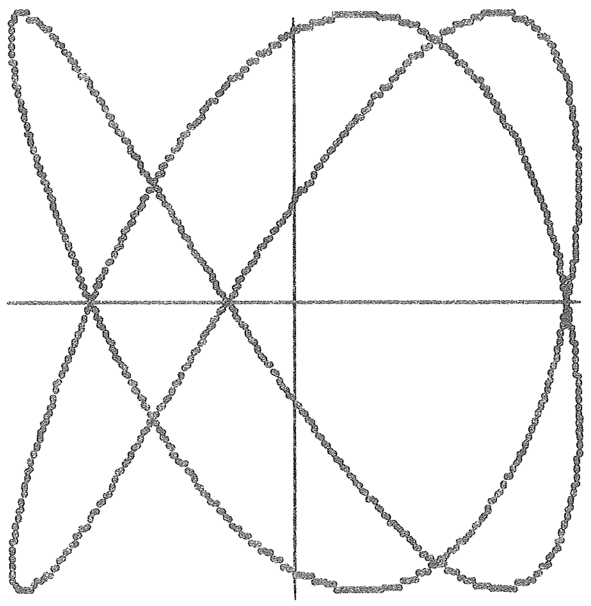
On this and the following pages are printed a number of examples of use of the plot program. Among them it is hoped that an example appears for most common situations. Every feature described above is illustrated at least once in either the plots which follow or the examples in the text above.

This document and all sample plots were produced on an AJ-832 terminal. The horizontal precision is 60 increments to the inch. The vertical precision is 48 increments to the inch.

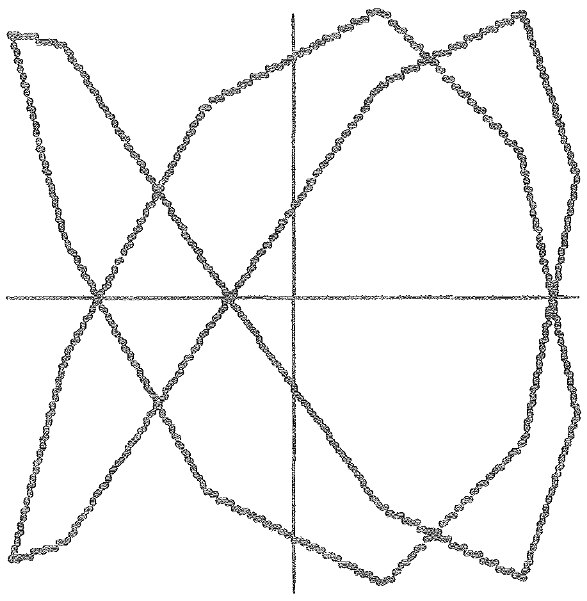
-------------------------||-------------------------

The plot on the title page of this document, reproduced below, was produced with the following input:

```
.PS 3 3
range x=-3:3 y=-3:3
define Lissajous(t):
        x = 3 * cos(2*t);
        y = 3 * cos(3*t + 2);
plot Lissajous(t) t=0:6.3 dt=.02
.PE
```

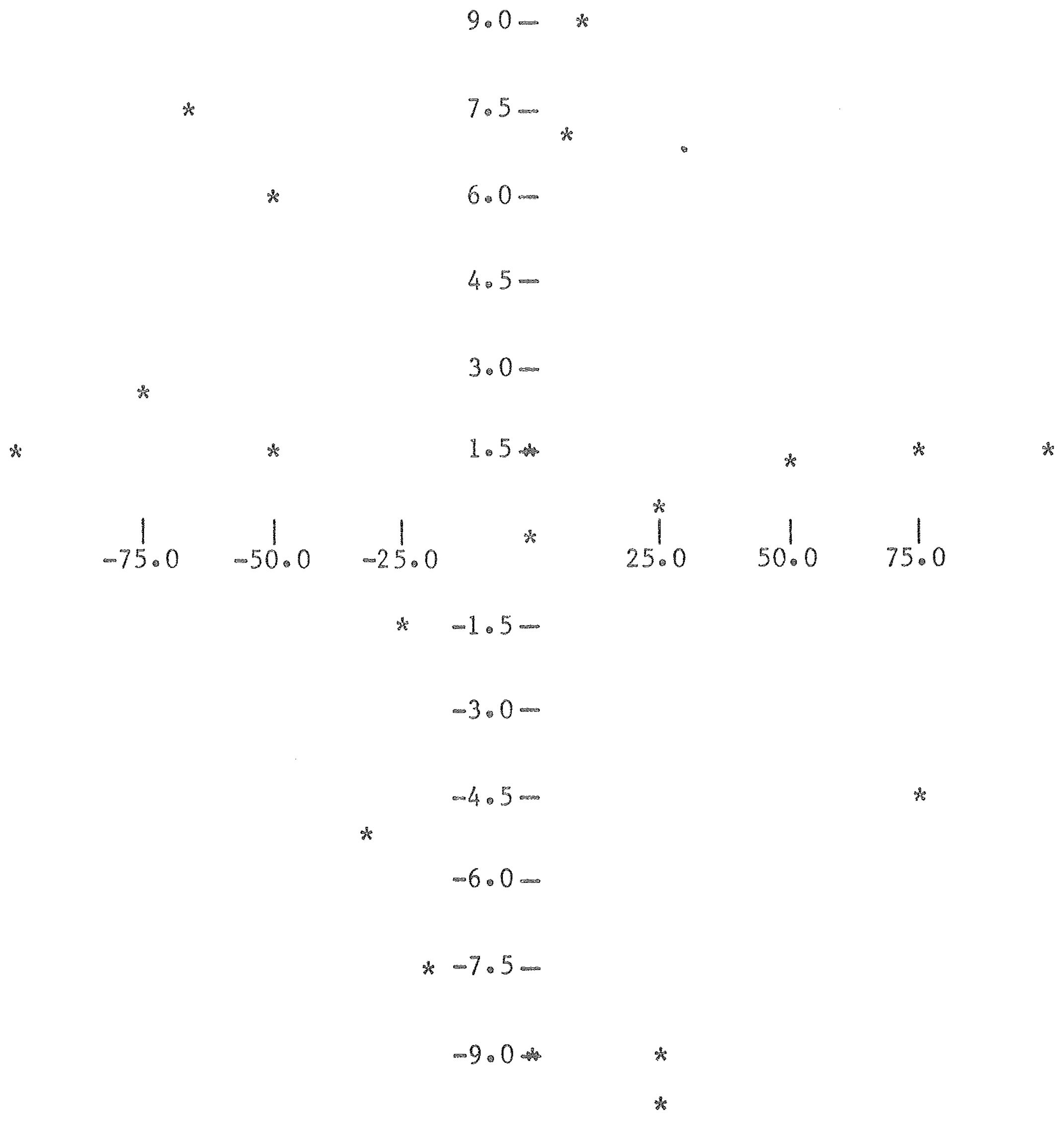


Notice the value given for dt in the plot statement. A relatively small value such as .02 will give one a smooth curve. An example of what the output looks like for a value of dt which is probably too large appears on the next page.



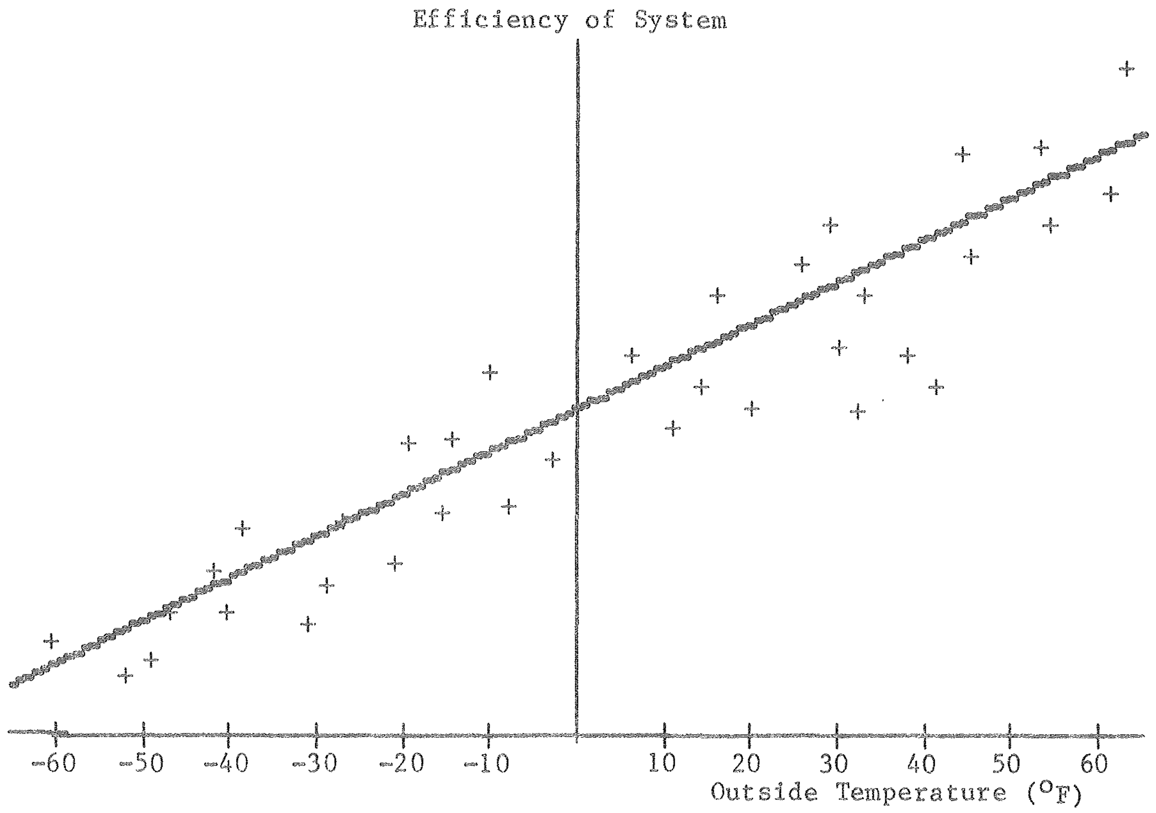
```
.PS 3 3
range x=-3:3 y=-3:3
define Lissajous(t):
        x = 3 * cos(2*t);
        y

 PE
```

E. SAMPLE PLOTS AND DESCRIPTIONS



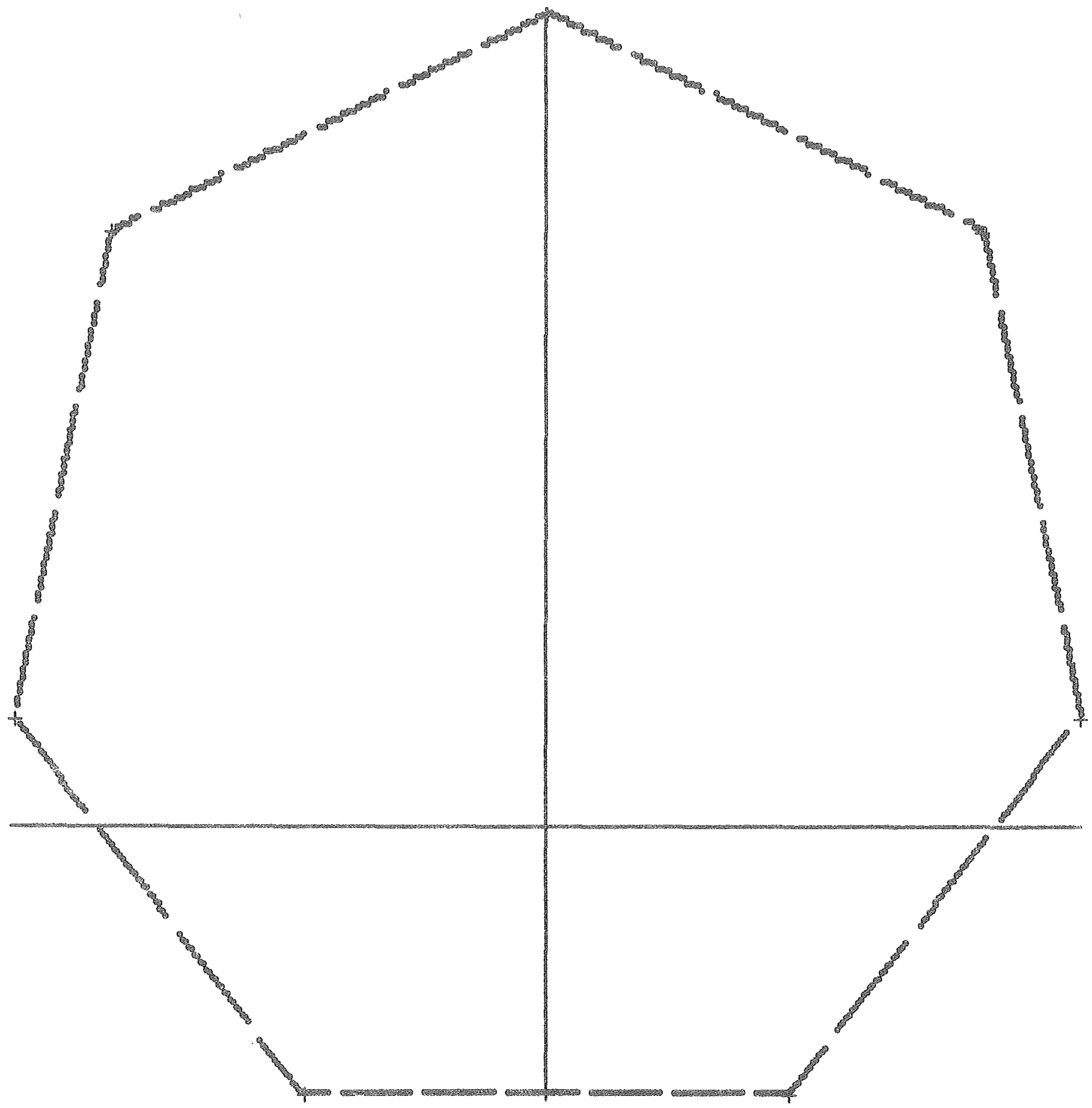
```
.PS 5.5 6 0
range x=-100:100 y=-10:10
marking x 25.0
marking y 1.5 left
define Test_Set : (-75,2.5) (25,-9) (50,1.3) (10,9) (-20,-7.5) (-25,-1.5)
                  (75,-4.5) (-50,1.5) (-32,-5.2) (25,.5) (-66,7.5)
                  (100,1.5) (0,-9) (-100,1.5) (7,7) (75,1.5) (-50,6) (0,1.5)
                  (25,-10) (0,0)
plot Test_Set "*"
noplot xy
.PE
```

Note that there was no centering performed due to the zero third argument on the ".PS" command.

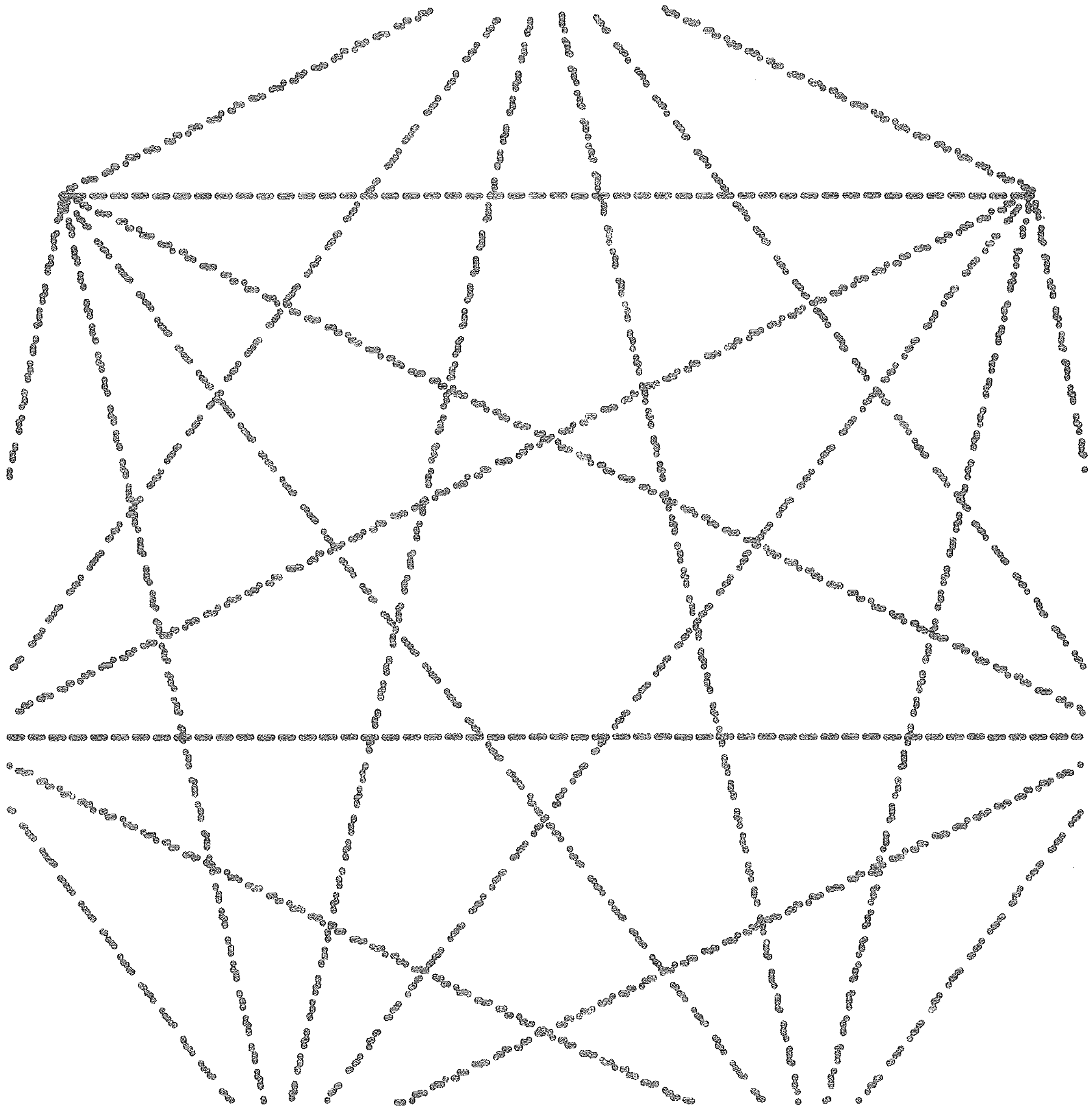


```
.PS 6 4
range x=-65:65 y=0:7
marking x 10
label x "Outside Temperature (\(deF)"
label y "Efficiency of System"
define Samplings : (-60.5,1) (-52,.6) (-49,.8) (-47,1.25) (-42,1.65)
        (-40.5,1.25) (-38.5,2.1) (-31,1.15) (-29,1.55) (-27,2.2)
        (-21,1.75) (-19.5,2.95) (-15.5,2.25) (-14.5,2.97) (-8,2.3)
        (-3,2.8) (-10,3.65) (6,3.8) (11,3.1) (14,3.5) (16,4.4)
        (20,3.3) (25.5,4.7) (29,5.1) (30,3.9) (32,3.25) (33,4.4)
        (38,3.8) (41,3.5) (44,5.8) (45,4.8) (53,5.9) (54,5.1)
        (61,5.4) (63,6.7)
define Linear : (65,6) (-65,.5)
plot Samplings
plot connect Linear
.PE
```

E. SAMPLE PLOTS AND DESCRIPTIONS



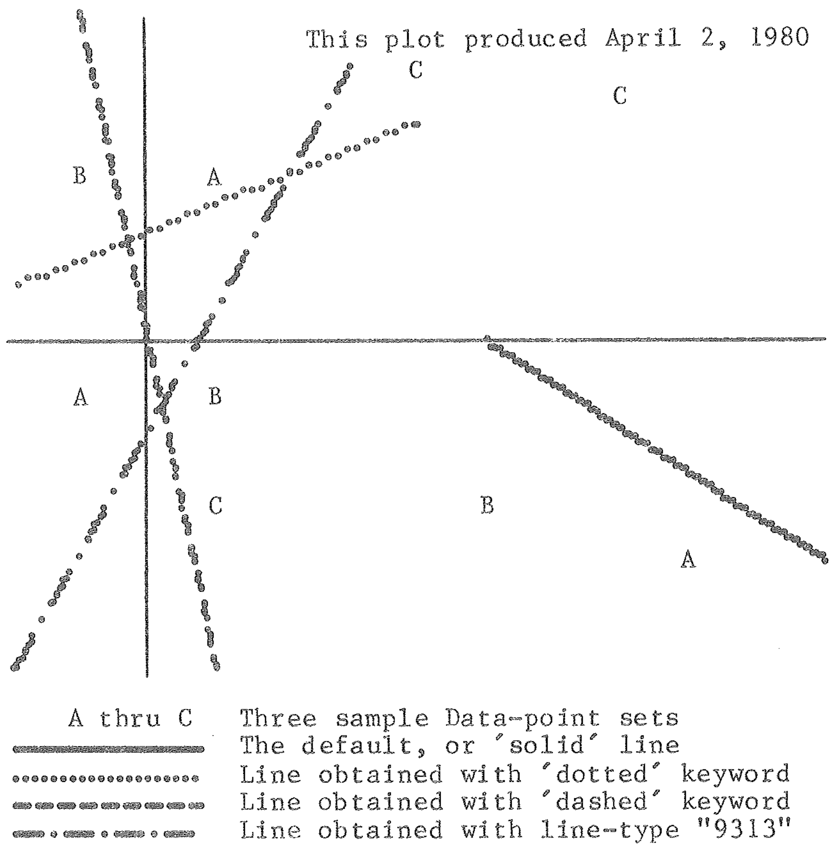
```
.PS 6 6
define heptagon : (0,15) (9,11) (11,2) (5,-5) (-5,-5) (-11,2) (-9,11) (0,15)
plot connect heptagon "909064"
plot heptagon
.PE
```



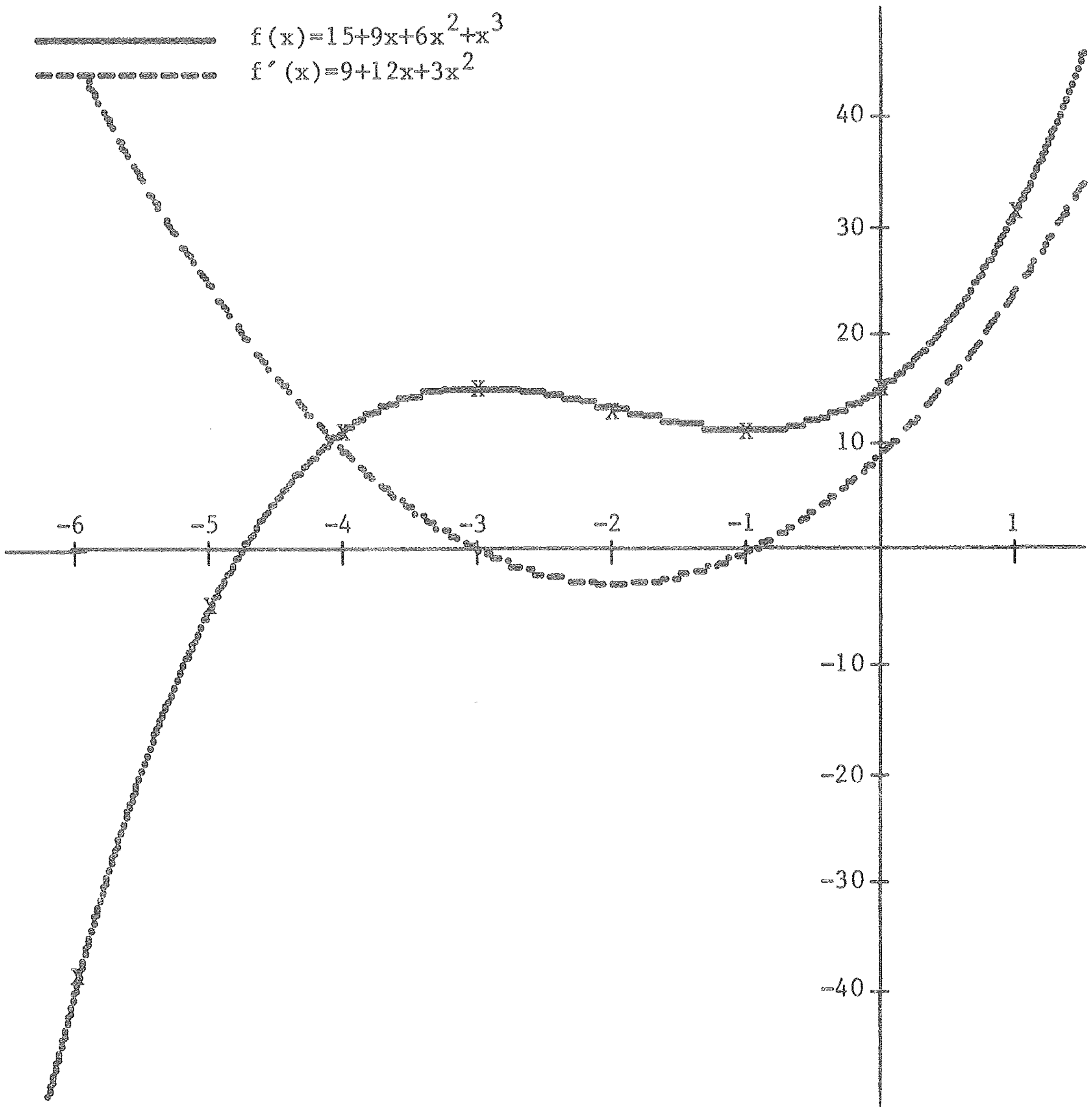
```
.PS 6 6
range x=-10:10 y=-4:14
define spirograph : (0,15) (9,11) (11,2) (5,-5) (-5,-5) (-11,2) (-9,11)
    (0,15) (11,2) (-5,-5) (-9,11) (9,11) (5,-5) (-11,2) (0,15) (5,-5)
    (-9,11) (11,2) (-11,2) (9,11) (-5,-5) (0,15)
plot connect spirograph dashed
noplot xy
.PE
```

Note that if parts of a plot lie outside the viewport, they will not be printed. That part of the plot which _is_ within the boundaries of the viewport will appear unaffected.

E. SAMPLE PLOTS AND DESCRIPTIONS

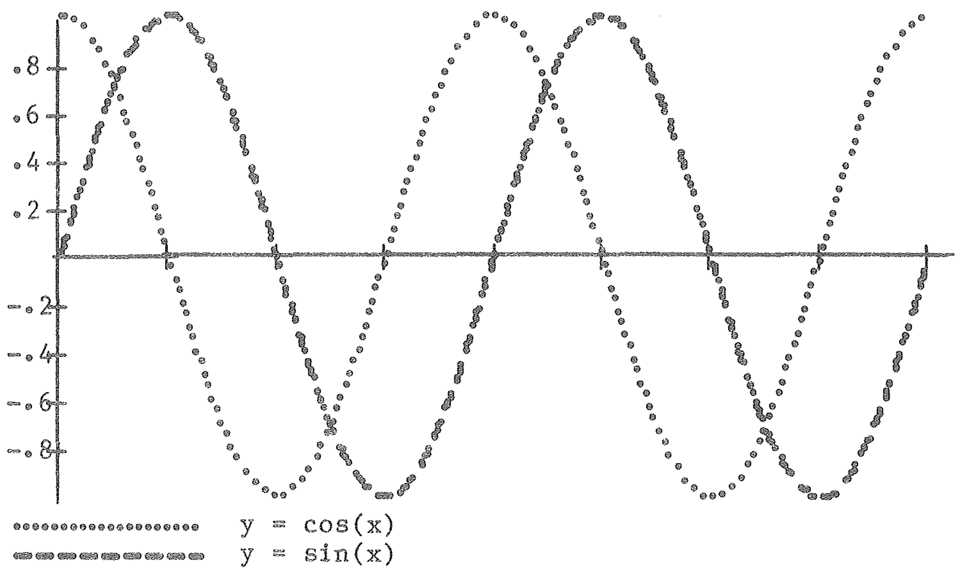


```
.PS 4.3 4.3
range x=-2:10 y=-6:6
viewport (0,.2) (1,1)
define a : (5,0) (10,-4)
define b : (4,4) (-2,1)
define c : (-1,6) (1,-6)
define d : (-2,-6) (3,5)
define setA : (1,3) (8,-4) (-1,-1)
define setB : (-1,3) (5,-3) (1,-1)
define setC : (1,-3) (4,5) (7,4.5)
plot connect a solid
plot connect b dotted
plot connect c dashed
plot connect d "9313"
plot setA A
plot setB B
plot setC C
legend l(0,.133) 'A thru C' "Three sample Data-point sets"
legend l(0,.1)   solid      "The default, or 'solid' line"
legend l(0,.066) dotted     "Line obtained with 'dotted' keyword"
legend l(0,.033) dashed     "Line obtained with 'dashed' keyword"
legend l(0,0)    "9313"     "Line obtained with line-type \"9313\""
write  l(2.5,5.5)  "This plot produced \*(DY"
.PE
```

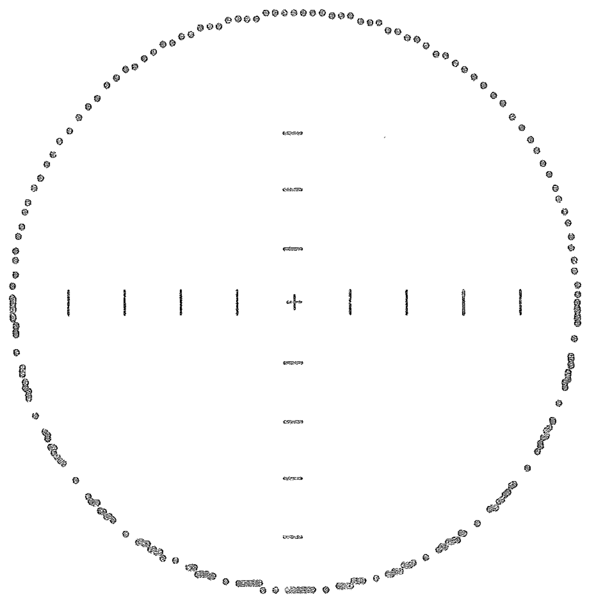


```
.PS 6 6
range x=-6.5:1.5 y=-50:50
marking x 1 top
marking y left 10
define cubic(x) = 15 + 9*x + 6*x**2 + x**3;
define diff(x) = 9 + 12*x + 3*x**2;
define points : (-6,-39) (-5,-5) (-4,11) (-3,15) (-2,13)
                (-1,11) (0,15) (1,31) (2,65)
plot cubic(x)
plot diff(x) dashed x=-5.9:1.5
plot points "X"
legend (-6.3, 47) solid "f(x)=15+9x+6x\u2\d+x\u3\d"
legend (-6.3, 43.5) dashed "f′(x)=9+12x+3x\u2\d"
.PE
```

E. SAMPLE PLOTS AND DESCRIPTIONS



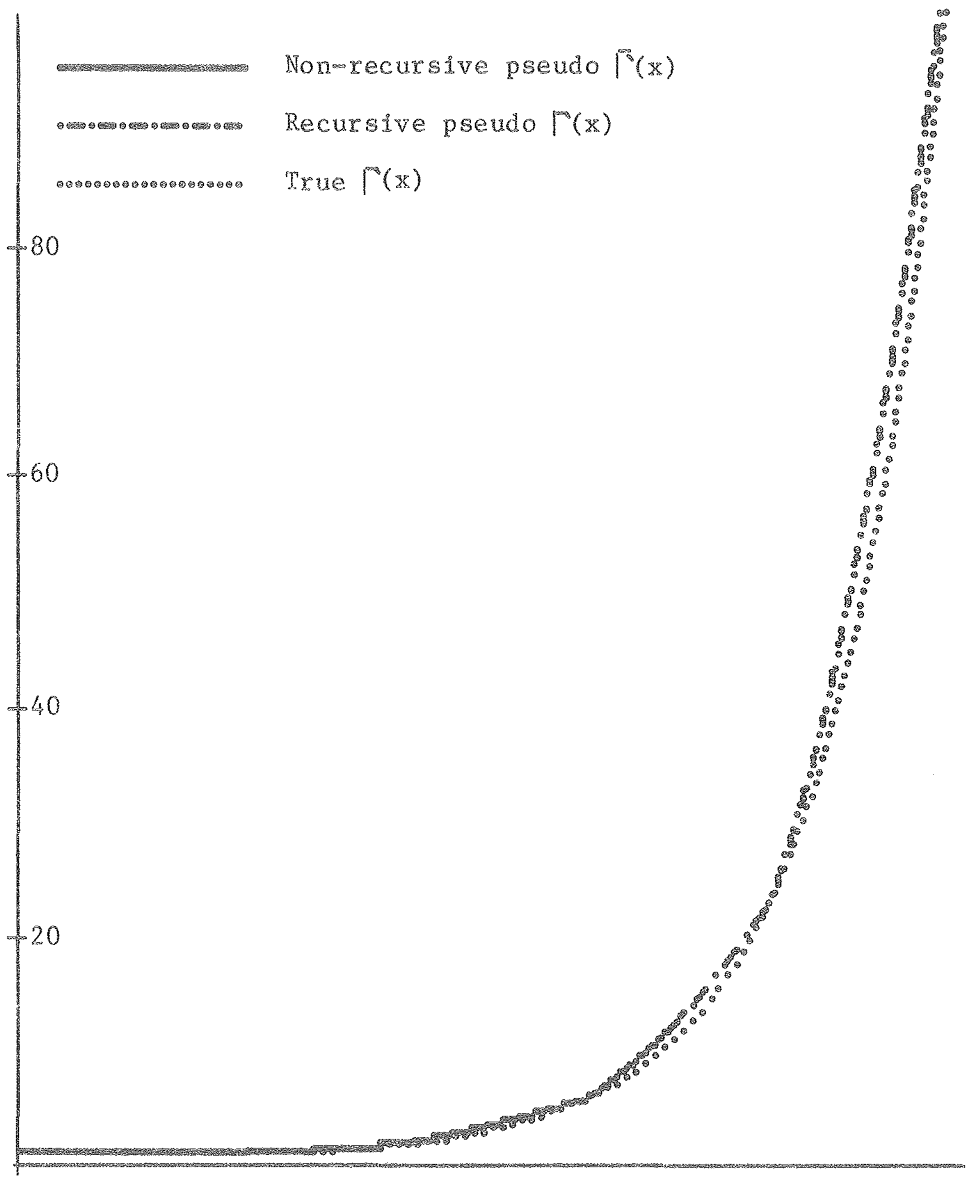
```
.PS 5 3
range x=0:13
viewport (0.0833,.166) (1.666,1)
marking x 1.57 nonumber
marking y .2 left
define f(x) = sin(x);
define g(x) = cos(x);
plot f(x) x=0:12.56 dashed
plot g(x) x=0:12.56 dotted
legend L(0,.0555) dashed "y = sin(x)"
legend L(0,.1111) dotted "y = cos(x)"
.PE
```



```
.PS 3 3 1
range x=-1:1 y=-1:1
marking xy .2 nonumber
define c1(x) = sqrt(1 - x**2);
define c2(x) = -c1(x);
define center : (0,0)
plot c1(x) dotted
plot c2(x) "9313"
plot center
noplot xy
.PE
```

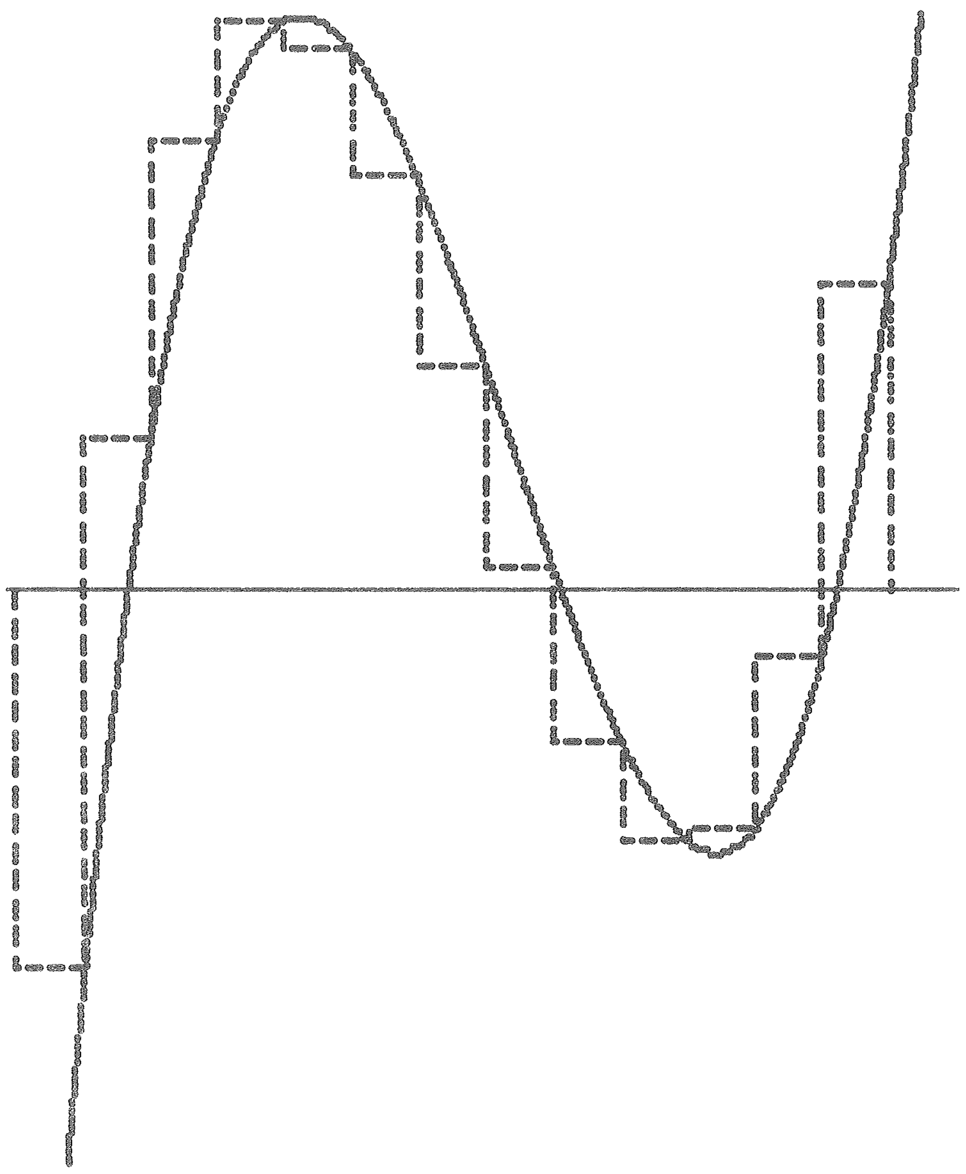
Notice the definition of c2(x) which includes a reference to an earlier-defined function, c1(x). A function-definition may always reference to a previously defined user function. In fact, a function may even be recursive (i.e. reference itself) as in the next example.

E. SAMPLE PLOTS AND DESCRIPTIONS

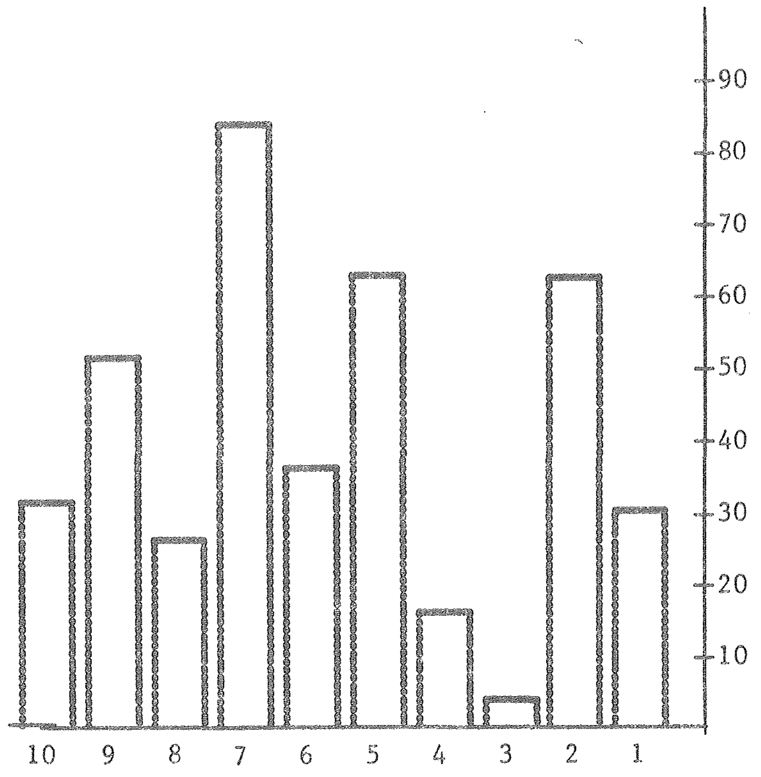


```
.PS 4 5
range x=0:5 y=0:100
marking y 20
define factor1(x) :
        prod = 1;
    10: if (x <= 1) goto 99;
        prod = prod * x;
        x = x - 1;
        goto 10;
    99: factor1 = prod;
        end;
define factor2(x) :
        if (x <= 1) goto 10;
        factor2 = x * factor2(x - 1);
        goto 99;
    10: factor2 = 1;
    99: end;
define factor3(x) = gamma(x+1);
plot factor1(x) x=0:3.5 solid
plot factor2(x) x=3.5:5 "1252"
plot factor3(x) dotted
legend (.2,95) solid "Non-recursive pseudo \(*G(x)"
legend (.2,90) "1252" "Recursive pseudo \(*G(x)"
legend (.2,85) dotted "True \(*G(x)"
.PE
```

E. SAMPLE PLOTS AND DESCRIPTIONS

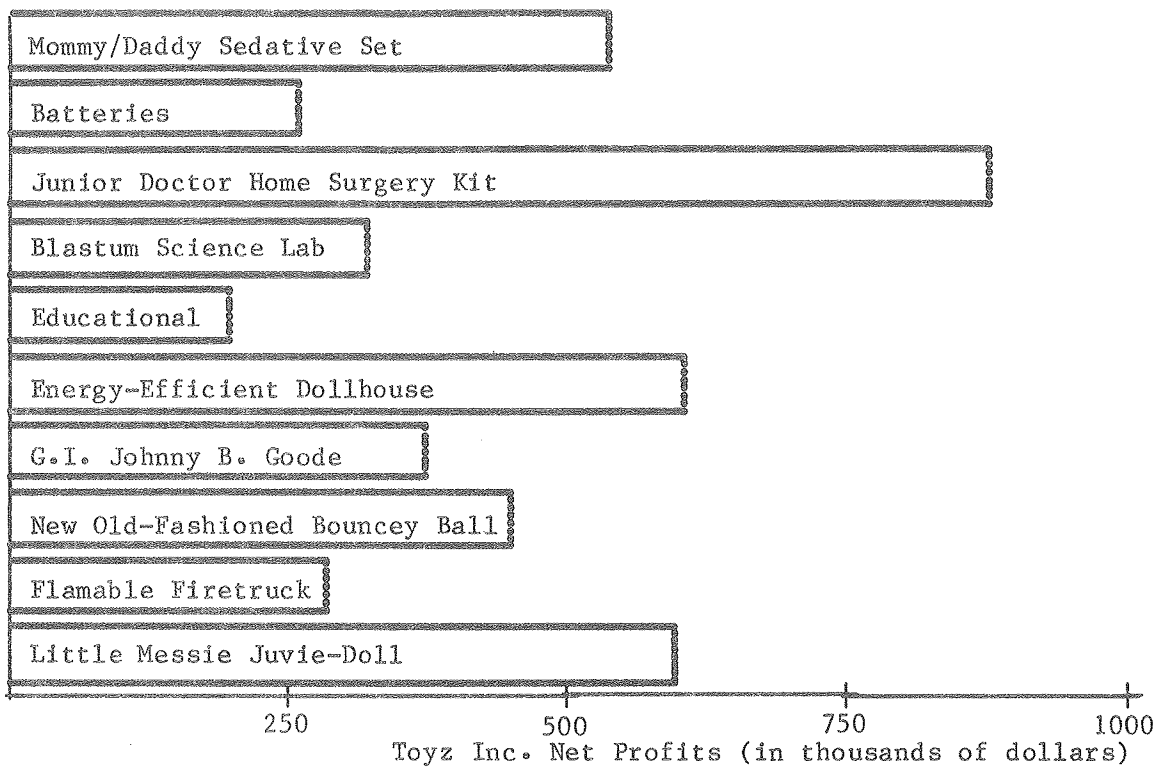


```
.PS 5 6
range x=-8.1:6 y=-80:80
define points : (-7,-53) (-6,21) (-5,63) (-4,79) (-3,75) (-2,57)
                (-1,31) (0,3) (1,-21) (2,-35) (3,-33) (4,-9)
                (5,43)
define function(x) = x**3 + 2*x**2 - 27*x + 3;
plot histogram points dashed
plot function(x)
noplot y
.PE
```

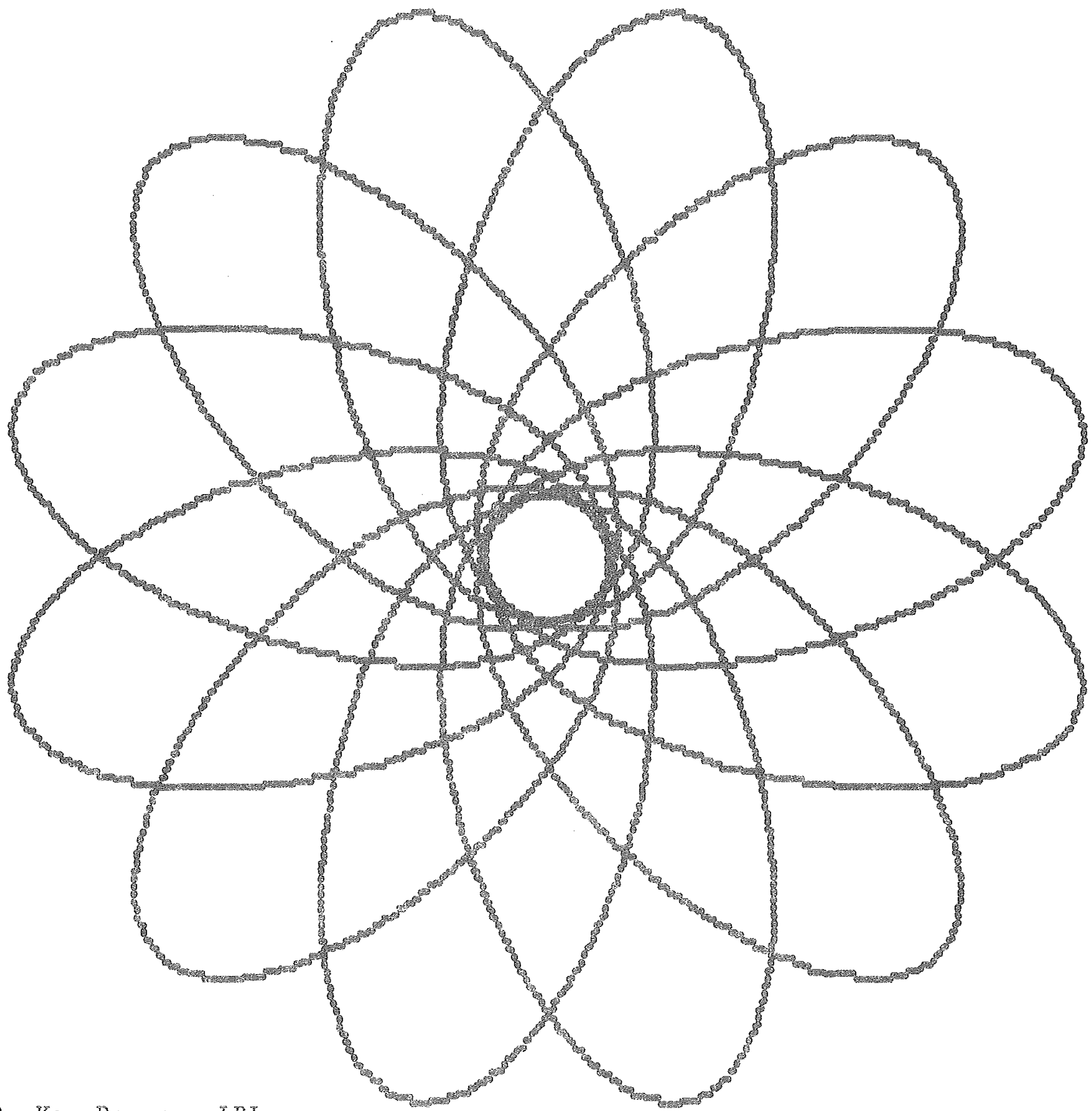


```
.PS 4 4
range x=10.5:0 y=0:100
marking x 1 nohash
marking y 10
define Sample_barg : (0.6,30) (1.6,62) (2.6,4) (3.6,16) (4.6,63) (5.6,36)
                         (6.6,84) (7.6,26) (8.6,51) (9.6,31)
plot  bargraph  Sample_barg  width=.8  vertical
.PE
```

E. SAMPLE PLOTS AND DESCRIPTIONS



```
.PS 6 4
range x=0:1010 y=0:10
marking x 250
label x "Toyz Inc. Net Profits (in thousands of dollars)"
define gross_profit : (594,1) (284,2) (450,3) (374,4) (603,5)
    (195,6) (320,7) (876,8) (259,9) (537,10)
plot bargraph gross_profit
write (25,.5) "Little Messie Juvie-Doll"
write (25,1.5) "Flamable Firetruck"
write (25,2.5) "New Old-Fashioned Bouncey Ball"
write (25,3.5) "G.I. Johnny B. Goode"
write (25,4.5) "Energy-Efficient Dollhouse"
write (25,5.5) "Educational"
write (25,6.5) "Blastum Science Lab"
write (25,7.5) "Junior Doctor Home Surgery Kit"
write (25,8.5) "Batteries"
write (25,9.5) "Mommy/Daddy Sedative Set"
.PE
```



By Ken Revzan, LBL

E. SAMPLE PLOTS AND DESCRIPTIONS

```
.PS 6 6
range x=-9:9 y=-9:9
define alpha(x) = 12 * x;
define rosette (t) :
    x :
        term1 = (5 * cos(alpha(t)) - 4)  *  cos(t);
        term2 = 3 * sin(alpha(t)) * sin(t);
        x = term1 + term2;
        end;
    y = -(5 * cos(alpha(t)) - 4)  *  sin(t) +
                                     3 * sin(alpha(t)) * cos(t);
plot rosette (t) t=0:6.3  dt=.003
write 1(0,0) "By Ken Revzan, LBL"
noplot xy
.PE
```